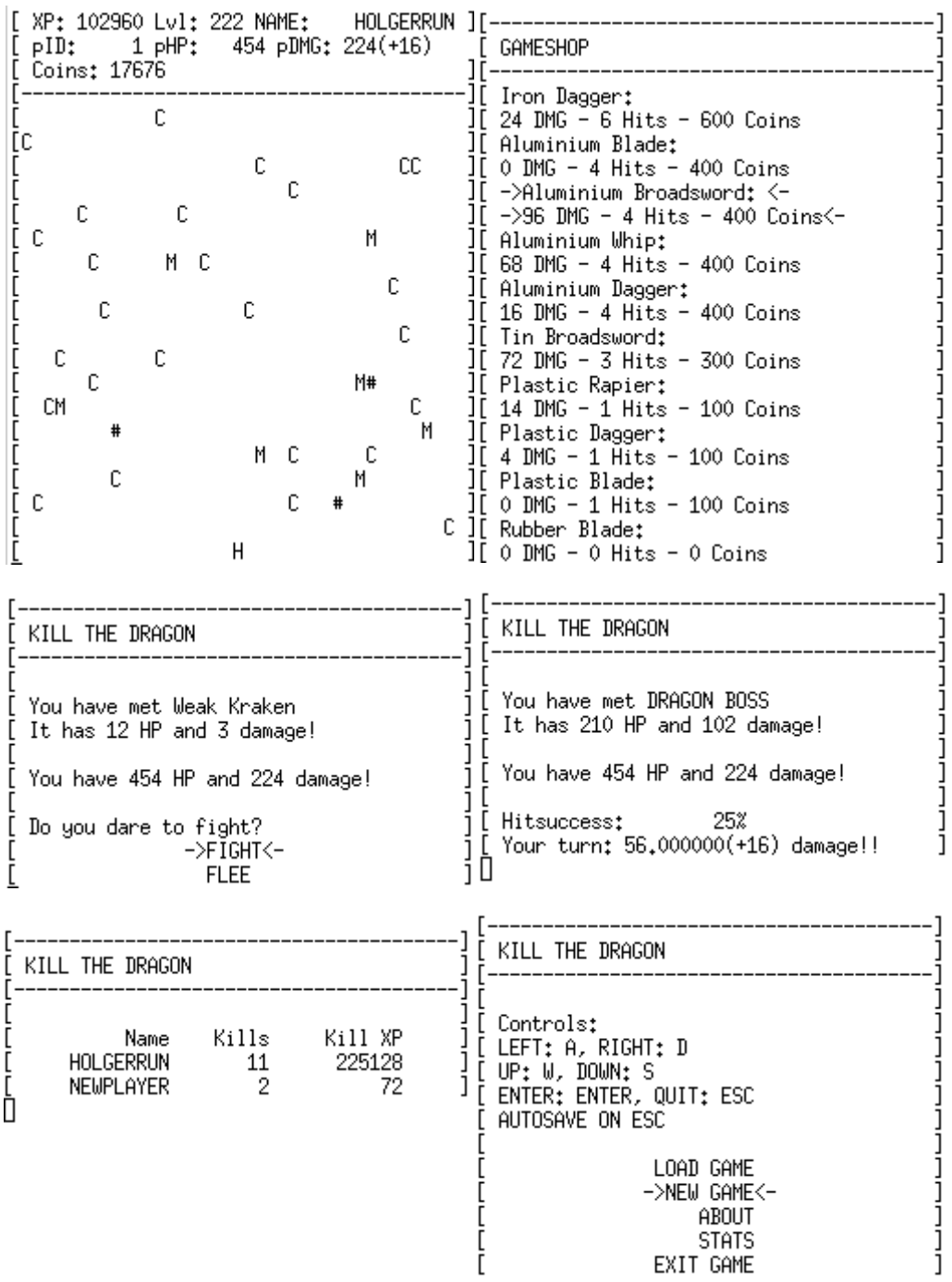
Kill the dragon



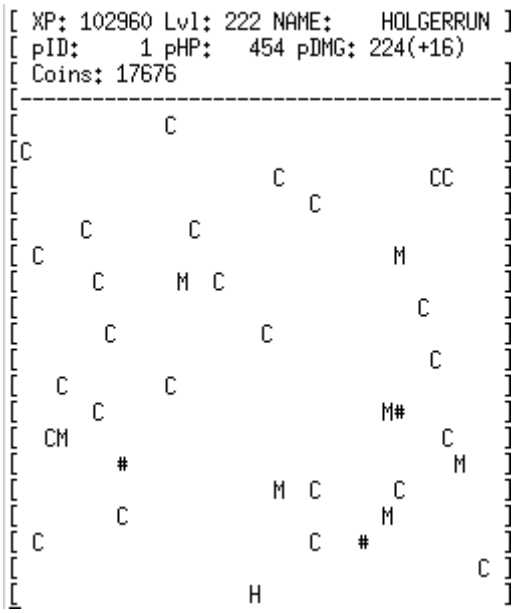
Indhold

Beskrivelse af system 3
1. iteration 3
Frontend 3
Backend 4
2. iteration 4
Frontend 4
Backend 5
3. iteration 5
Frontend 5
Backend 5
Use cases 6
1. iteration 6
2. iteration 7
3. iteration 7
Use case diagram 7
1. iteration 7
2. iteration 8
3. iteration 8
Domain model 9
1. iteration 9
2. iteration 10
3. Iteration 11
Class diagram 12
1. iteration 12
2. iteration 13
3. iteration 14
Sequence diagrams 15
1. iteration 15
2. iteration 16

3. Iteration 18

SQL 19

Beskrivelse af system



1. iteration

Frontend

Du er en Hero som bevæger dig igennem en verden. Verden består af bogstaver du kan interagere med. H'et er heroen(dig) og du kan bevæge dig til højre og venstre ved at bruge A og D på tastaturet. Du kan kun dø en gang, så bliver din hero slettet.

Et bogstav du kan finde er M for monster - Det er en fjende som du kan slå ihjel. Du kan også flygte fra dem hvis de er for stærke, men pas på! Når først du har taget kampen op så kan du ikke flygte længere.
Når du har dræbt en fjende, så modtager du en masse XP for det. Det hjælper dig til at stige i level.

Et andet bogstav er D for dragon - Det er også et monster, og endbossen. Hvis du dræber den her, så vinder du spillet. Du kan dog ikke flygte fra den, så vent med at engage den her, hvis ikke du har lyst til at starte forfra.

Et tredje bogstav er C for coin - Det er ingen fjende, men i stedet så giver den health og strength. Den kan også hjælpe dig med at stige i level, hvis fjenderne er for stærke til at dræbe. Du får mere XP for coins desto mere du stiger i level.

Du kan altid lave en ny hero og heroen gemmer automatisk i en SqliteDB når du trykker ESC fra spillet.

Backend

Main består af en masse menuer. Herudover opretter main en instans af character og et monster efter behov. Disse har ikke nødvendigvis noget med hinanden at gøre.

Characteren bliver oprettet når man laver en hero eller loader en hero.
Verden er et array af strings, som bliver generet i world klassen.
Når main er i game menuen så hiver den hele tiden en ny linje ud af verden. Dette får verden til at bevæge sig og får world klassen til at generere nye linjer.
Characteren skifter position ved at trykke på A og D som får et instans af keyboard control til at sætte heroens position. - De har ikke noget direkte med hinanden at gøre.
En thread i main tjekker om nogen har trykket på tastaturet og hvis man har det, så udfører den en opgave.

Hvis heroen rammer et bogstav i verden, så sker der ting, der er defineret i main. Hvis man samler coins op, så får heroen XP og HP. HP kun intil det er makimalt udfyldt.

Hvis man rammer et monster. Dette inkludere en drage. Så skifter spillet menu. Her bliver lavet et instans af et monster som man så kan slås med i en anden menu. Hvis man slås med en drage, så får man ikke muligheden for at flygte.

Monstrene, ikke inkluderet dragen, får et autogeneret navn baseret på hvor stærke de er og en random race.

Når man har slået monsteret ihjel så får man noget XP. Så kan man bruge coins til at generere ens HP.

Spillet gemmer og loader fra en Sqlite DB igennem SQLdatabase klassen.
SQLdatabasen har alt ansvaret for SQL.

2. iteration

Frontend

Spillet er nu blevet udvidet med caves. I cavesene kan man finde mange monstre af gangen. Det betyder at man kan få meget mere XP. Derudover så er der nu Guld med i spillet. Du får guld for at fjerne alle monstrene fra de caves der er. Caves er makeret med #.

I caves er der op til 5 monstre. De kan ikke være dragen, men alle andre. De kan max blive level 100 individuelt. Så hvis du når level 500, er du basicly udødelig, hvis du bare husker at heale.

Backend

De nye caves bliver genereret fra caveFactory som har kontrollen over hvordan caves bliver lavet.

Derudover så er monsterklassen også bygget om og der er kommet en monsterFactory klasse som styrer hvor monstre bliver produceret.

Det betyder også at caveFactory kalder monsterFactory. Caves og Monster ”friender” begge to deres factory klasser fordi de så har adgang til at sætte deres private værdier. Disse klasser og deres factoryklasser er altså derfor ret forbundet, men de ejer ikke hinanden.

Når man rammer en cave, så kalder cavefactory, monsterfactory og når cavefactory får monstrene af monsterfactory, så opretter den et array af fjender, som man så kan se om man tør kæmpe imod. Caven får så monstrene. Caven ejer altså ret meget monstrene. Hvis caven forsvinder, så forsvinder monstrene også. Der er altså et ret stærkt forhold her.
Gør man det, kæmper man mod et monster, individuelt, til caven er tom. Når man tømmer en cave for fjender, hvis man ikke dør i forsøget, så får man en masse guld, baseret på cavens lvl. Hertil også ekstra XP ud over det man får for at dræbe monstrene individuelt.

3. iteration

Frontend

Nu kan du bruge dit guld til noget. Der er blevet oprettet en shop hvor du kan købe våben i startmenuen. Det kræver dog du loader en karakter først.
Du generere våben ved at tømme caves, hvor du nu får/finder et brugt våben som hurtigt går i stykker. Våbnets styrke afhænger af hvor stærk caven er.
Når først en hero har fundet et nyt våben, bliver det tilgængeligt for alle heroes.

Derudover når en hero laver et kill eller tømmer en cave, så bliver det gemt i et særligt stats table hvor det bliver muligt at se hvor mange man har dræbt og hvor meget XP man har fået ud af det pr hero.

Backend

Nu er der også oprettet end weapon class og en weaponFactory class. Weapons bliver oprettet i caves, hvor de bliver genereret baseret på caves lvl. Når weaponFactory generere våben, sender den en forespørgsel til SqlDB om at lave et nyt våben, hvis

våbnet allerede eksisterer gør SqlDB ikke noget yderligere.
Det er altså weaponFactory som beder SqlDB om at oprette våben i SQL. De hænger altså sammen, men tyndt.

Der er også oprettet en ny menu i hovedemenuen hvor man kan shoppe våben som er genereret. Her spørger main, SqlDB om hvilke våben som eksisterer. SqlDB returnere nu selve våbene som er genereret op i en vektor. På den måde hænger Weapon klassen også sammen med SqlDB. Dog lidt tyndt.

Characteren har våbnet stored i sin klasse. Ergo er der også noget komposition her. Hvis karakteren dør, så forsvinder våbnet også.

Derudover er der også lavet noget statistik i denne iteration, så hver gang man dræber et monster, så bliver SqlDB bedt om at adde killXP og killCount til et specielt Statistik table. På den måde så kan man stadig se heroens stats, selv efter han er død og er blevet slettet.

Dette table's ID er også heroens ID, så det er nemt at finde rundt i. Der kunne altså være en foreign key her, men da heroens ID kan blive slettet, har jeg ikke i SQL lavet det til en foreign key.

I SQL gemmer Heroes nu et WEAPON_ID hvis den har noget et weapon. Dette er en foreign key til WeaponsTypes. Derudover en WEAPON_HIT, hvilket er i forhold til weapon durability.

Der er også blevet lavet en statsEntry som bare er en struct for at gøre det nemt at loade alle disse data fra SQL over i en vektor.

Use cases

1. iteration

En **spiller** åbner spillet og opretter en bruger/"Hero".
Heroen bliver oprettet i en **Sqlite DB**.
Heroen starter med 10 health/liv og 5 skade.
Heroen starter i en lille **matrixverden**.
Heroens flytter sig til siderne for at imødegå **fjender**/**monstre** og **coins** som rykker ned fra toppen af skærmen. **Coins** giver **XP**, **health**/**liv** og **gold**. **Heroen** kan også slås med **monstre** herunder **dragen**. Alle **monstre** giver også XP. Heroen samler **XP** for at stige i **level**.
Når **heroen** har samlet sit **level** * 1000 i **XP** stiger den et **level**. Med nyt **level** får **heroen** 2 **health**/**liv** ekstra samt 1 **skadestyrke** ekstra.
Dragen er **bossen** og det er ikke nemt at dræbe den. **Heroen** kan flygte fra alle **monstre** undtagen **dragen**.

Både **fjenden** og **heroen** rammer ikke altid lige godt og derfor kan **heroen** godt få tæv af en lidt dårligere **fjende** eller vice versa.
Heroen kan kun dø en gang så bliver hans **bruger** slettet. Hvis **heroen** vinder over **dragen** har **heroen** vundet.
En **spiller** har brug for en pause, så han trykker på ESC for pausemenu. **Heroens** data bliver gemt i **DB**.
En **spiller** kommer tilbage for at komme videre i et gammelt spil, spillet kan findes under LOAD GAME

2. iteration

<<Udvidelse fra første>>

Heroen kan også gå ind i caves og kæmpe mod flere monstre. Dette giver **Gold** for at tømme caven.

3. iteration

<<Udvidelse fra anden>>

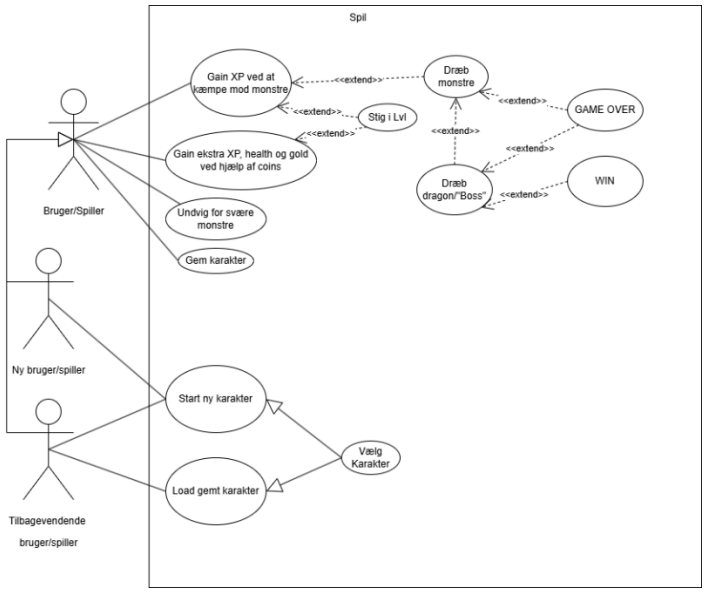
Heroen får en brugt version af et våben i caves når den har tømt caven.

Spilleren kan så købe en ny version af våbnet i alle dens **heroes.**

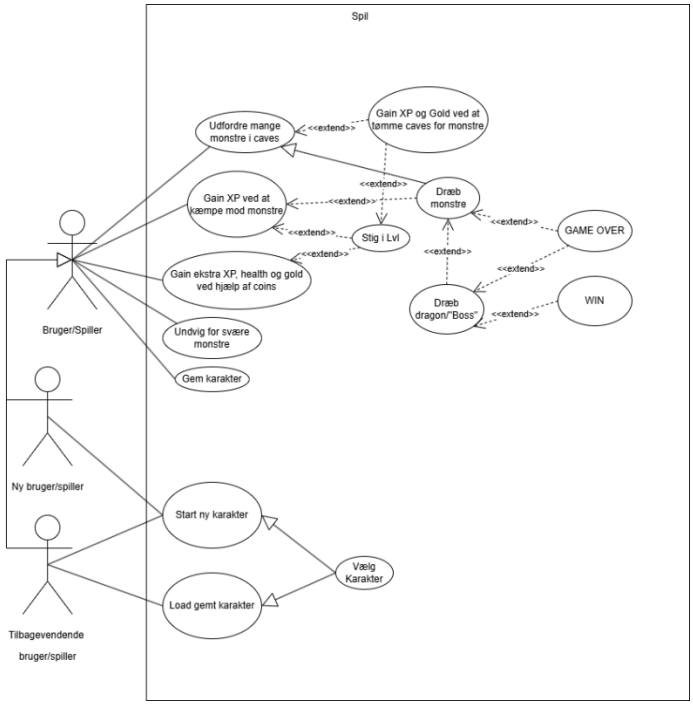
Hero's wins bliver gemt i statistik så man kan se hvem der er bedst.

Use case diagram

1. iteration



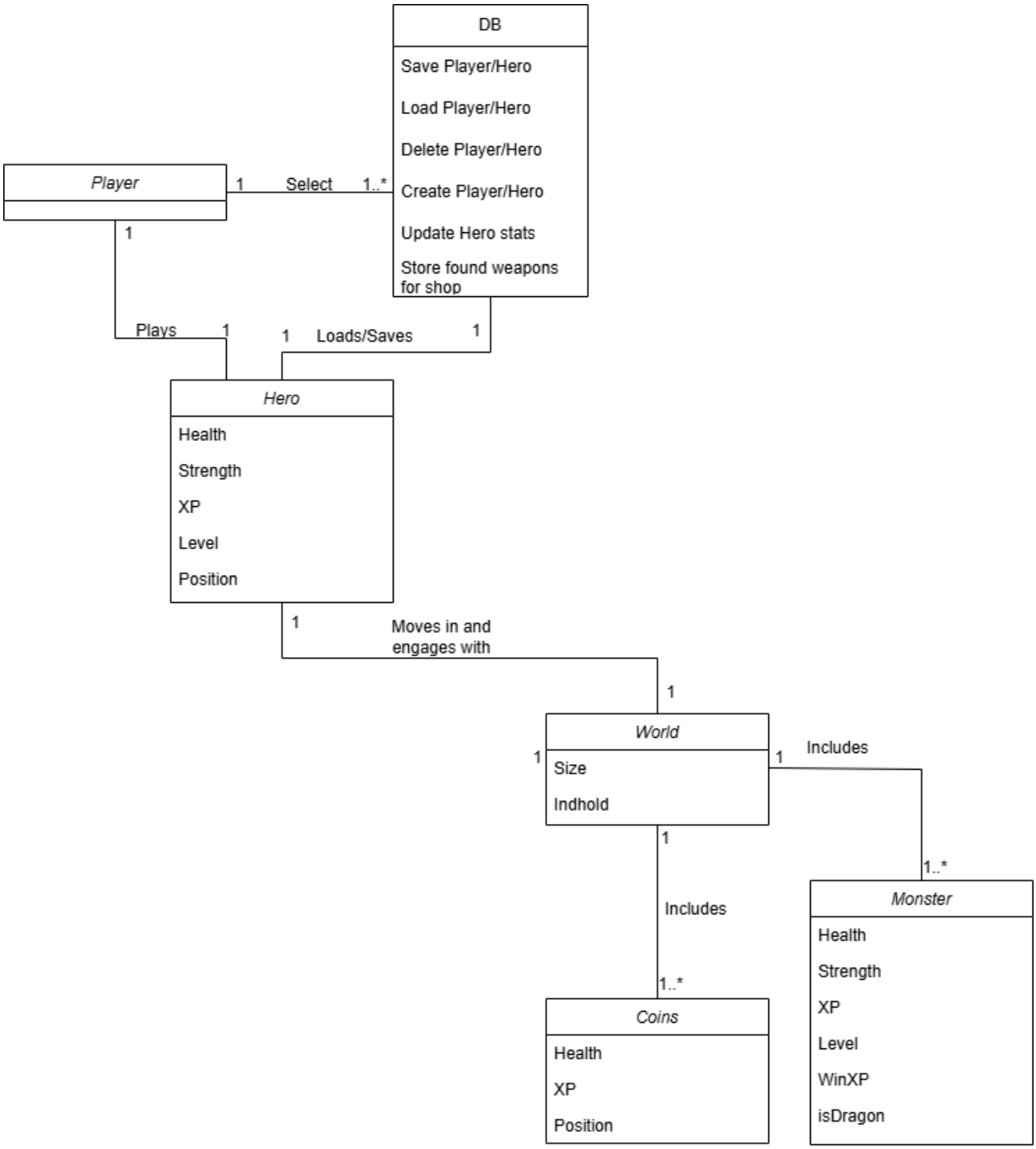
2. iteration



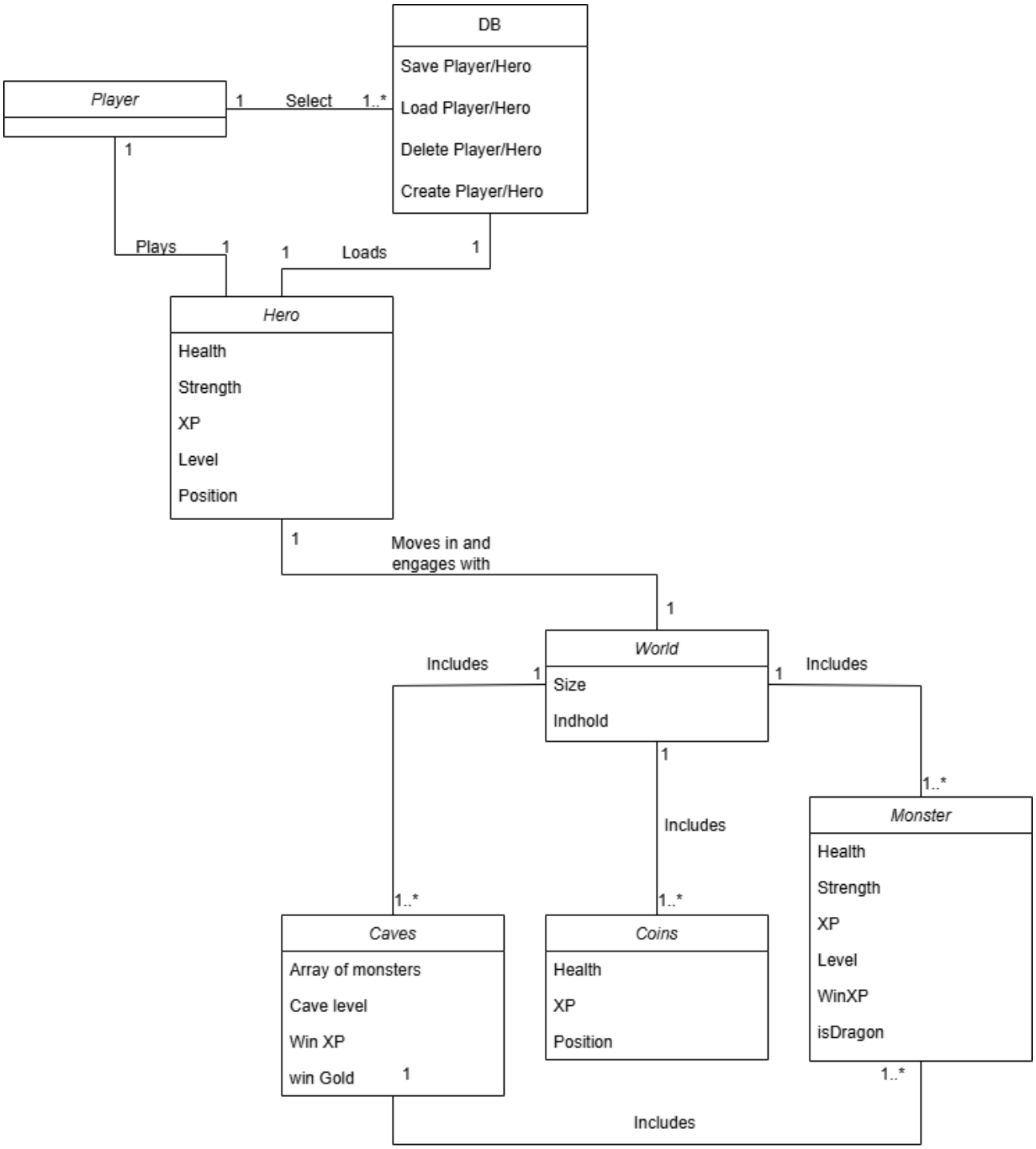
3. iteration

Domain model

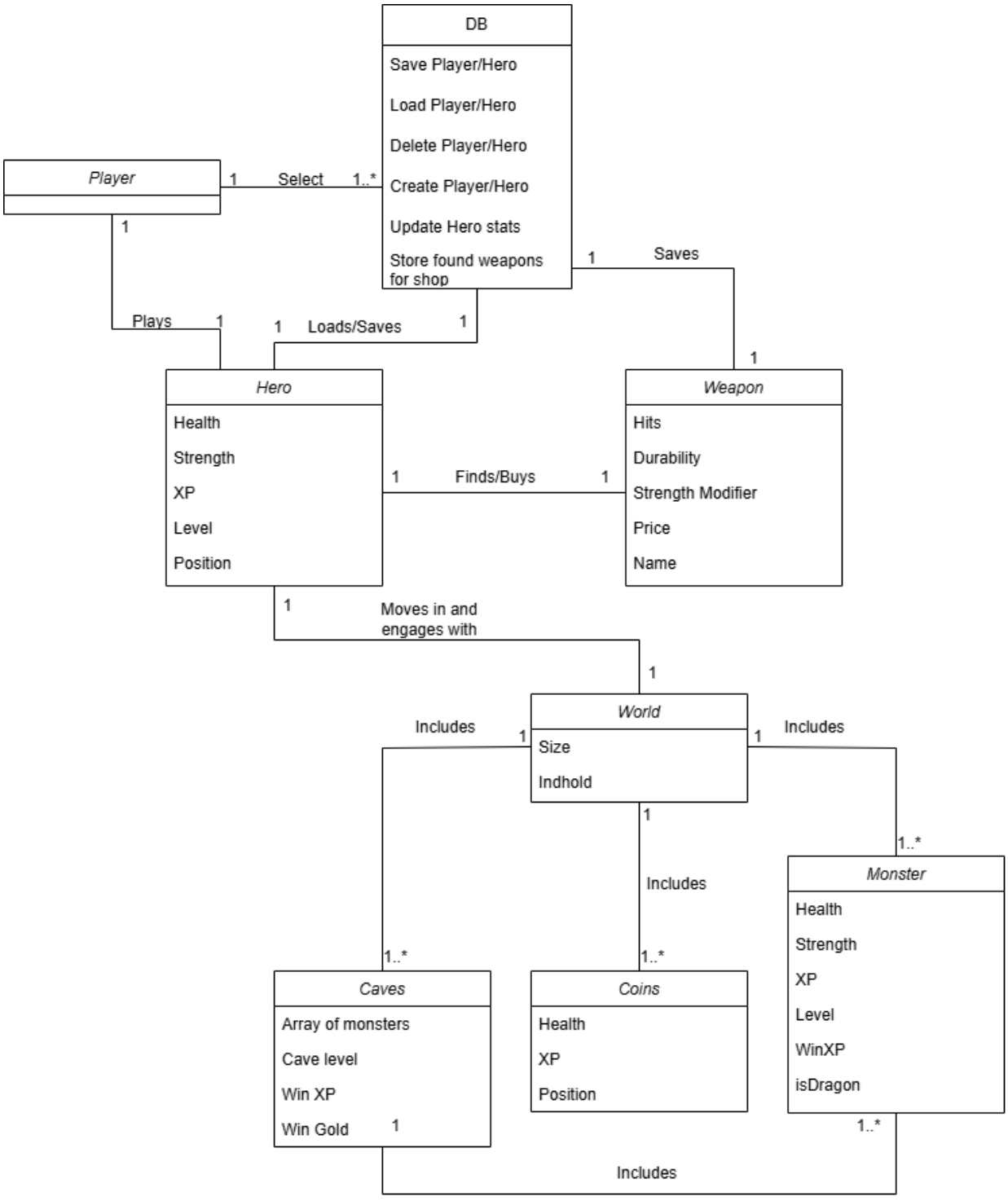
1. iteration



2. iteration


DB
Save Player/Hero
Load Player/Hero
Delete Player/Hero
Create Player/Hero
Player
1
Select
1..*
1
Plays
1
1
Loads
1
Hero
Health
Strength
XP
Level
Position
1
Moves in and
engages with
1
World
Size
Indhold
Includes
1
1
Includes
1
Includes
1..*
1..*
Monster
Health
Strength
XP
Level
WinXP
isDragon
Caves
Array of monsters
Cave level
Win XP
win Gold
Coins
Health
XP
Position
1
1..*
Includes

3. Iteration

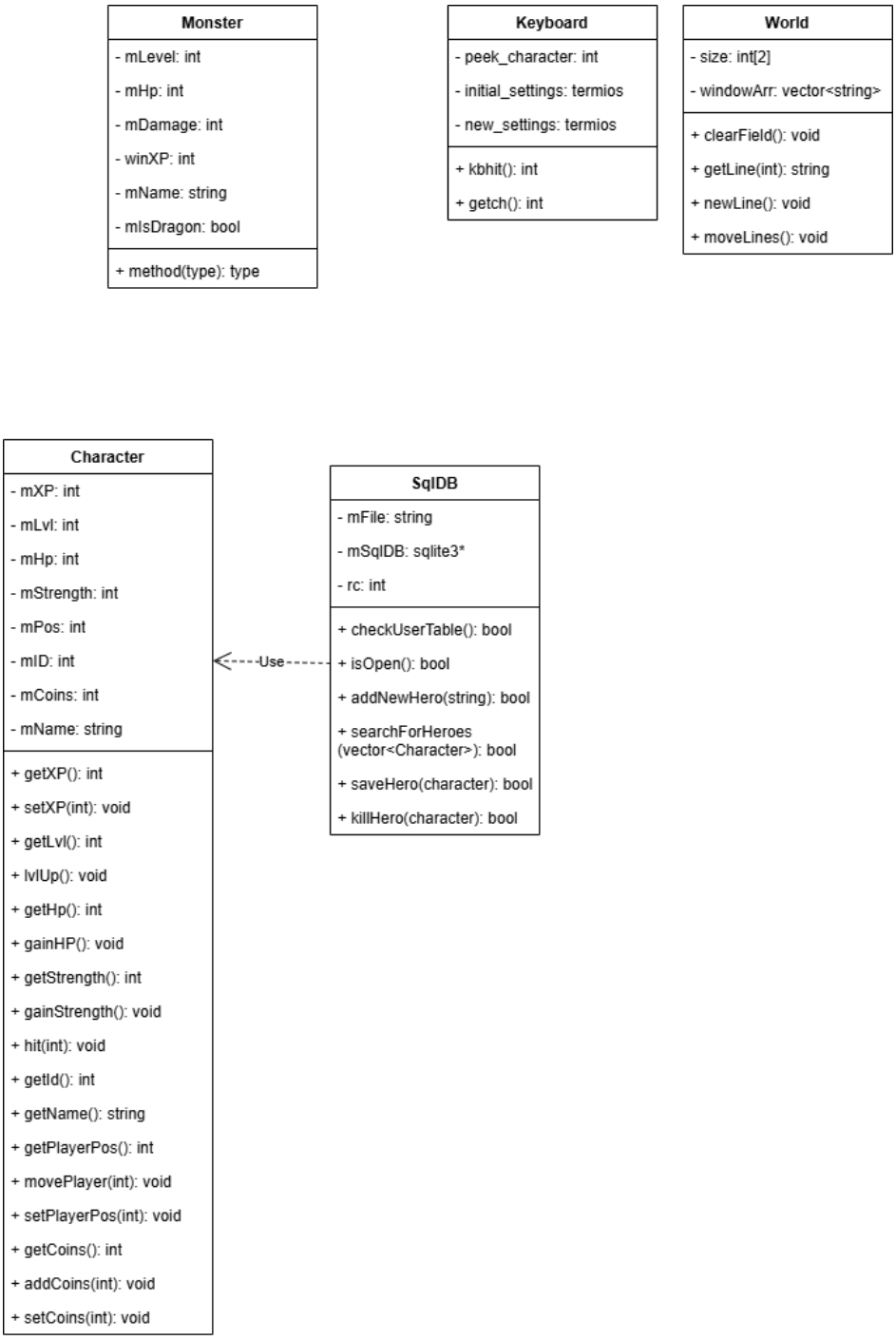

DB
Save Player/Hero
Load Player/Hero
Delete Player/Hero
Create Player/Hero
Update Hero stats
Store found weapons for shop
Player
1
Select
1..*
1
Saves
1
Plays
1
1
Loads/Saves
1
1
Hero
Health
Strength
XP
Level
Position
Weapon
Hits
Durability
Strength Modifier
Price
Name
1
Finds/Buys
1
1
Moves in and engages with
1
World
Size
Indhold
Includes
1
1
Includes
1
1..*
Includes
1..*
1..*
Caves
Array of monsters
Cave level
Win XP
Win Gold
1
Coins
Health
XP
Position
Monster
Health
Strength
XP
Level
WinXP
isDragon
1..*
Includes

Class diagram

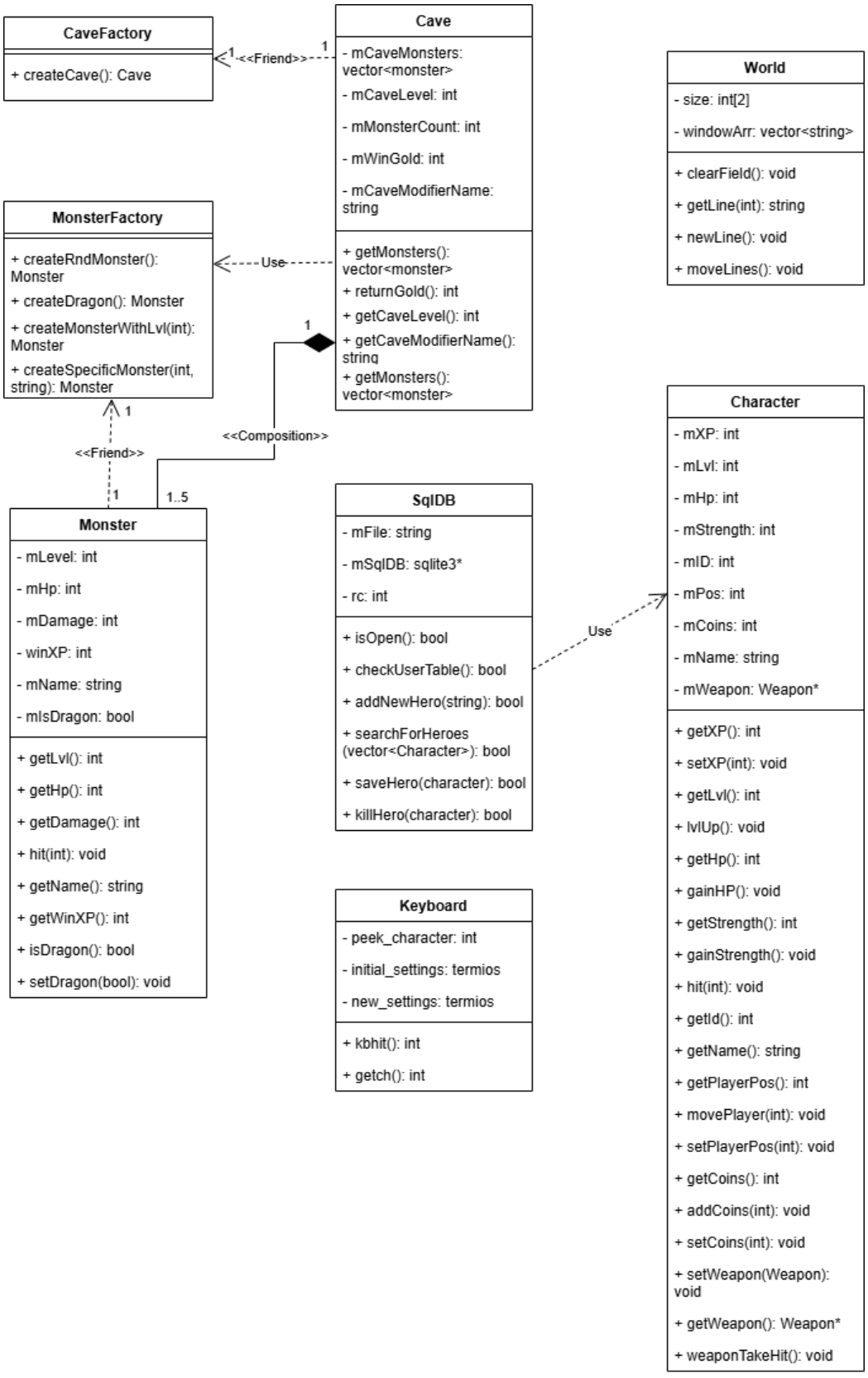
Fordi meget af formidlingen mellem klasserne er klaret i main er de ikke direkte forbundet. Det er getter og sætter metoder som bliver styret ude fra main.

World classen indeholder en masse spaces og specifikke bogstaver. De får først mening heroen rammer et bogstav i main. De er altså heller ikke connected. De kan eksistere uden hinanden whatsoever. Det er kun main der skaber mening imellem dem.

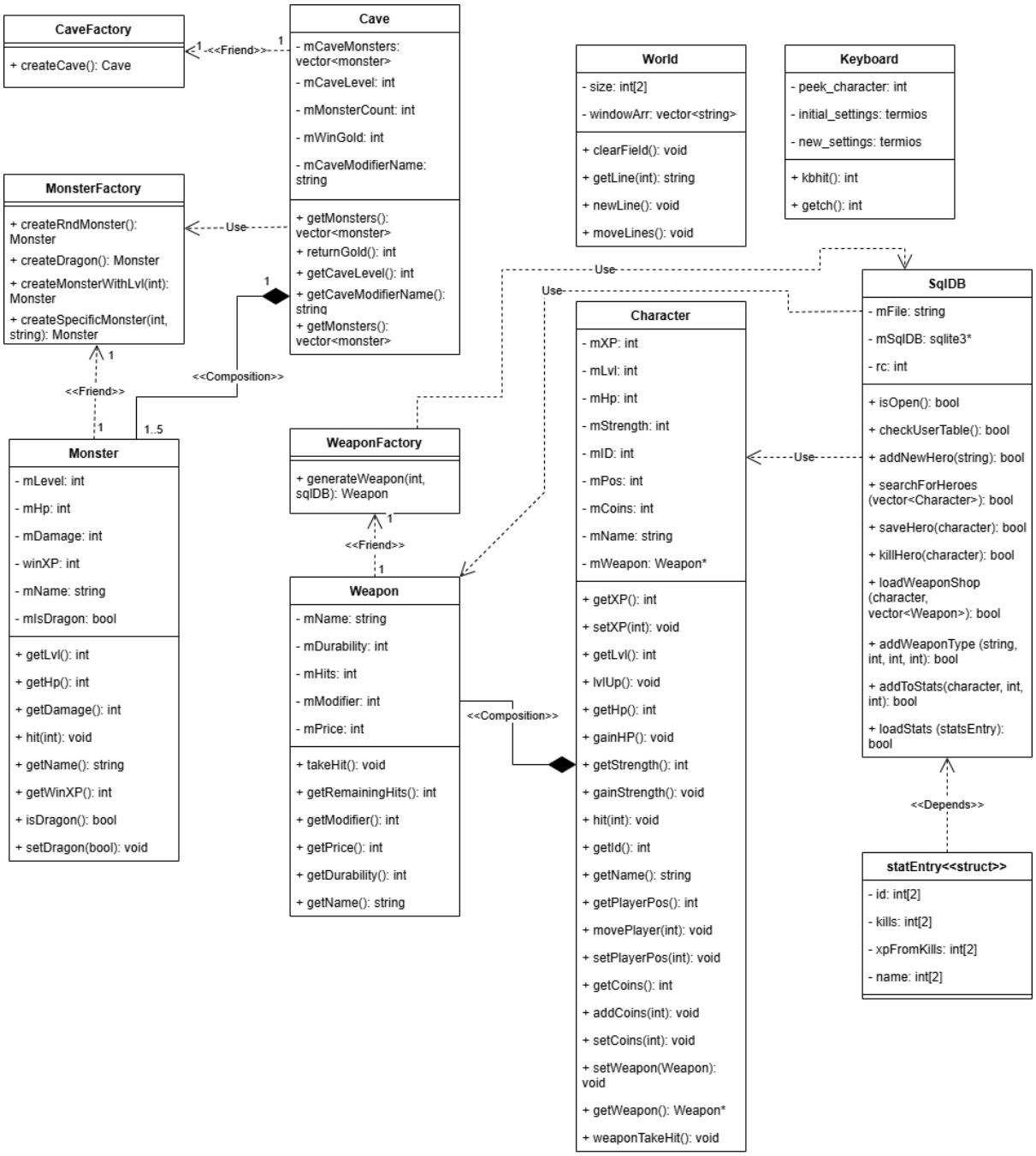
1. iteration



2. iteration


CaveFactory
+ createCave(): Cave
<<Friend>>
1
1
Cave
- mCaveMonsters: vector<monster>
- mCaveLevel: int
- mMonsterCount: int
- mWinGold: int
- mCaveModifierName: string
+ getMonsters(): vector<monster>
+ returnGold(): int
+ getCaveLevel(): int
+ getCaveModifierName(): string
+ getMonsters(): vector<monster>
World
- size: int[2]
- windowArr: vector<string>
+ clearField(): void
+ getLine(int): string
+ newLine(): void
+ moveLines(): void
MonsterFactory
+ createRndMonster(): Monster
+ createDragon(): Monster
+ createMonsterWithLvl(int): Monster
+ createSpecificMonster(int, string): Monster
Use
1
<<Composition>>
1
<<Friend>>
1
1..5
Monster
- mLevel: int
- mHp: int
- mDamage: int
- winXP: int
- mName: string
- mIsDragon: bool
+ getLvl(): int
+ getHp(): int
+ getDamage(): int
+ hit(int): void
+ getName(): string
+ getWinXP(): int
+ isDragon(): bool
+ setDragon(bool): void
SqlDB
- mFile: string
- mSqlDB: sqlite3*
- rc: int
+ isOpen(): bool
+ checkUserTable(): bool
+ addNewHero(string): bool
+ searchForHeroes (vector<Character>): bool
+ saveHero(character): bool
+ killHero(character): bool
Use
Character
- mXP: int
- mLvl: int
- mHp: int
- mStrength: int
- mID: int
- mPos: int
- mCoins: int
- mName: string
- mWeapon: Weapon*
+ getXP(): int
+ setXP(int): void
+ getLvl(): int
+ lvlUp(): void
+ getHp(): int
+ gainHP(): void
+ getStrength(): int
+ gainStrength(): void
+ hit(int): void
+ getId(): int
+ getName(): string
+ getPlayerPos(): int
+ movePlayer(int): void
+ setPlayerPos(int): void
+ getCoins(): int
+ addCoins(int): void
+ setCoins(int): void
+ setWeapon(Weapon): void
+ getWeapon(): Weapon*
+ weaponTakeHit(): void
Keyboard
- peek_character: int
- initial_settings: termios
- new_settings: termios
+ kbhit(): int
+ getch(): int

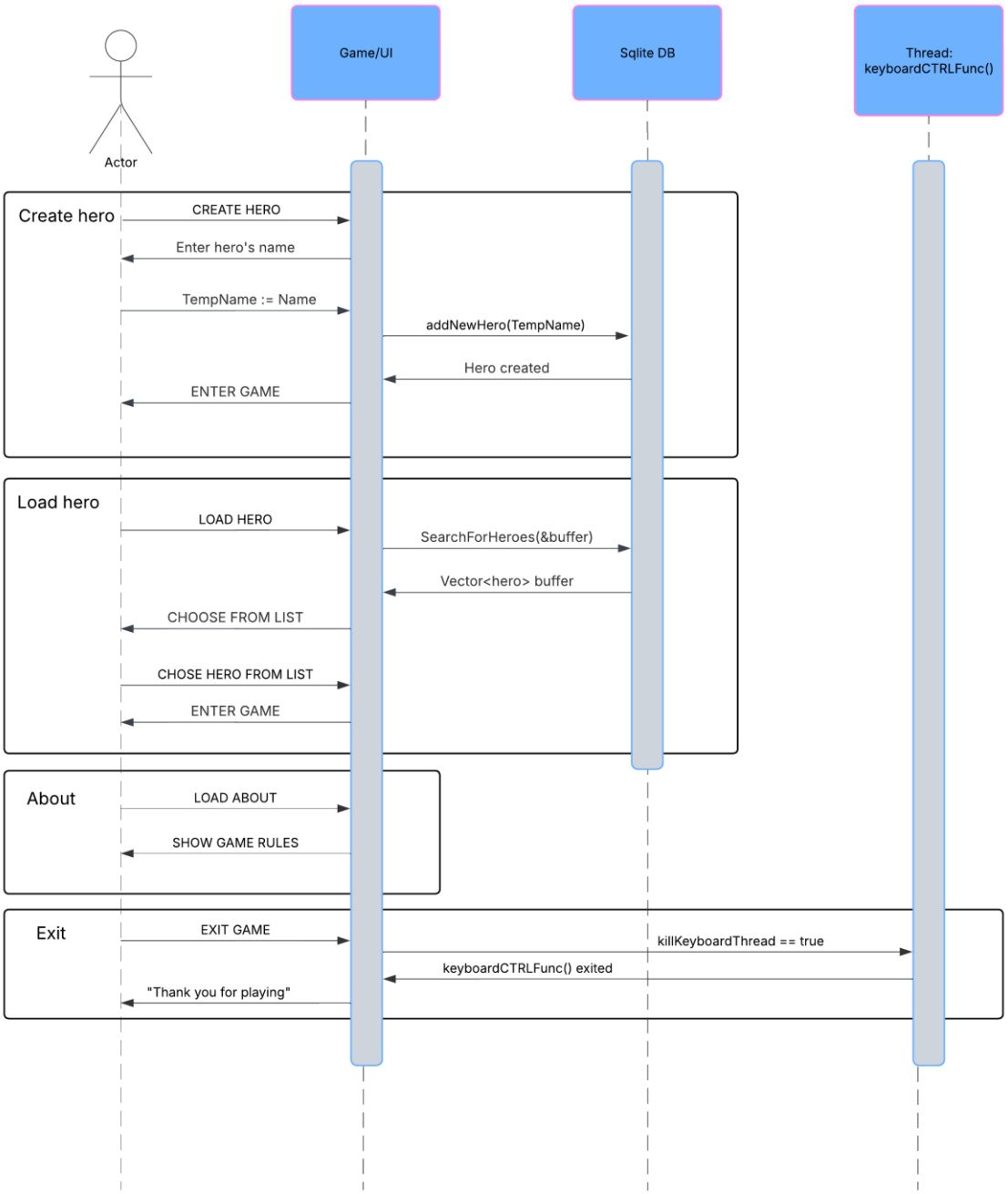
3. iteration


CaveFactory
+ createCave(): Cave
<<Friend>>
Cave
- mCaveMonsters: vector<monster>
- mCaveLevel: int
- mMonsterCount: int
- mWinGold: int
- mCaveModifierName: string
+ getMonsters(): vector<monster>
+ returnGold(): int
+ getCaveLevel(): int
+ getCaveModifierName(): string
+ getMonsters(): vector<monster>
World
- size: int[2]
- windowArr: vector<string>
+ clearField(): void
+ getLine(int): string
+ newLine(): void
+ moveLines(): void
Keyboard
- peek_character: int
- initial_settings: termios
- new_settings: termios
+ kbhit(): int
+ getch(): int
MonsterFactory
+ createRndMonster(): Monster
+ createDragon(): Monster
+ createMonsterWithLvl(int): Monster
+ createSpecificMonster(int, string): Monster
Use
<<Friend>>
<<Composition>>
Monster
- mLevel: int
- mHp: int
- mDamage: int
- winXP: int
- mName: string
- mIsDragon: bool
+ getLvl(): int
+ getHp(): int
+ getDamage(): int
+ hit(int): void
+ getName(): string
+ getWinXP(): int
+ isDragon(): bool
+ setDragon(bool): void
WeaponFactory
+ generateWeapon(int, sqlDB): Weapon
<<Friend>>
Weapon
- mName: string
- mDurability: int
- mHits: int
- mModifier: int
- mPrice: int
+ takeHit(): void
+ getRemainingHits(): int
+ getModifier(): int
+ getPrice(): int
+ getDurability(): int
+ getName(): string
<<Composition>>
Use
Character
- mXP: int
- mLvl: int
- mHp: int
- mStrength: int
- mID: int
- mPos: int
- mCoins: int
- mName: string
- mWeapon: Weapon*
+ getXP(): int
+ setXP(int): void
+ getLvl(): int
+ lvlUp(): void
+ getHp(): int
+ gainHP(): void
+ getStrength(): int
+ gainStrength(): void
+ hit(int): void
+ getId(): int
+ getName(): string
+ getPlayerPos(): int
+ movePlayer(int): void
+ setPlayerPos(int): void
+ getCoins(): int
+ addCoins(int): void
+ setCoins(int): void
+ setWeapon(Weapon): void
+ getWeapon(): Weapon*
+ weaponTakeHit(): void
Use
SqlDB
- mFile: string
- mSqlDB: sqlite3*
- rc: int
+ isOpen(): bool
+ checkUserTable(): bool
+ addNewHero(string): bool
+ searchForHeroes (vector<Character>): bool
+ saveHero(character): bool
+ killHero(character): bool
+ loadWeaponShop (character, vector<Weapon>): bool
+ addWeaponType (string, int, int, int): bool
+ addToStats(character, int, int): bool
+ loadStats (statsEntry): bool
<<Depends>>
statEntry<<struct>>
- id: int[2]
- kills: int[2]
- xpFromKills: int[2]
- name: int[2]

Sequence diagrams

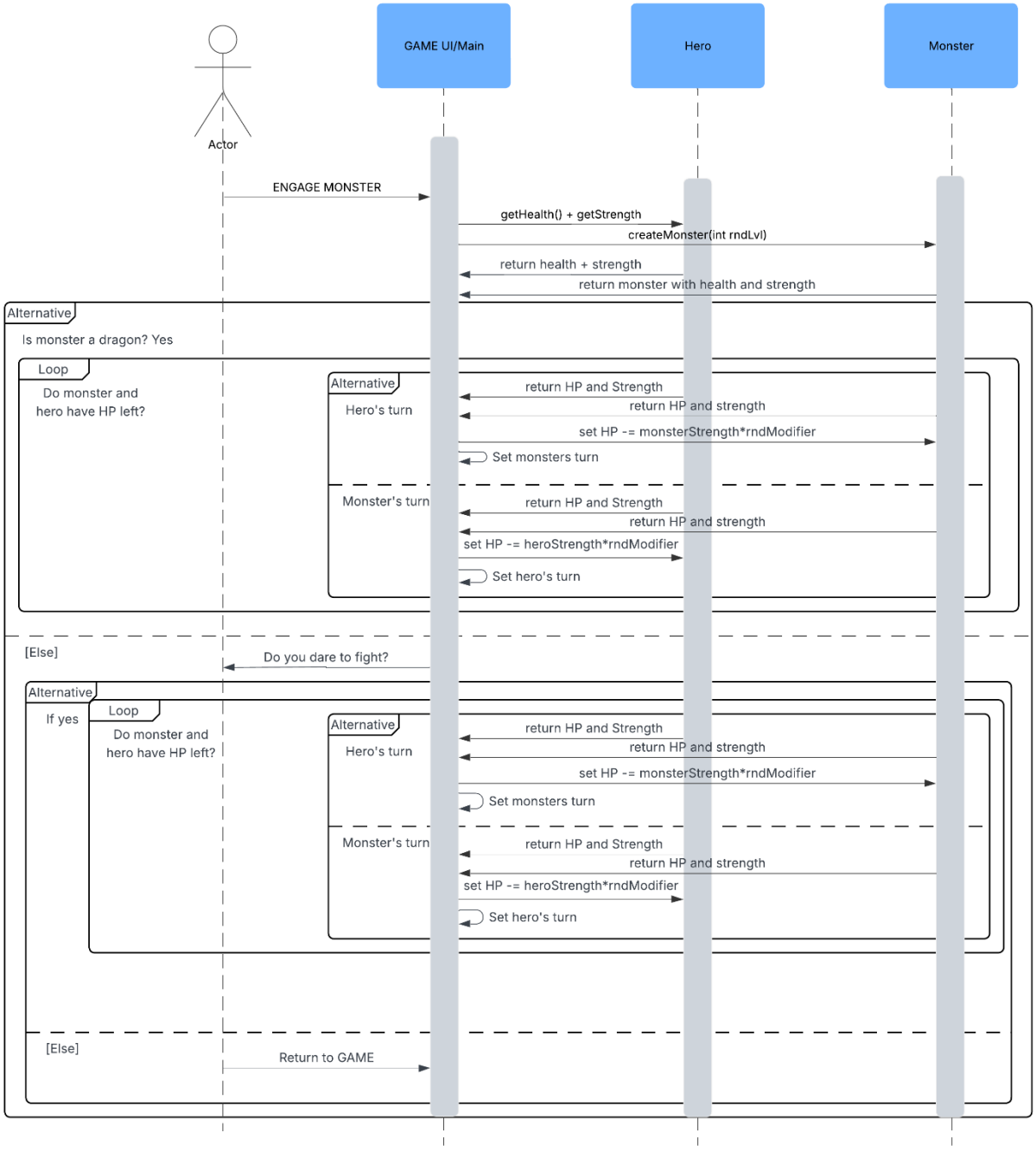
1. iteration

Player menu

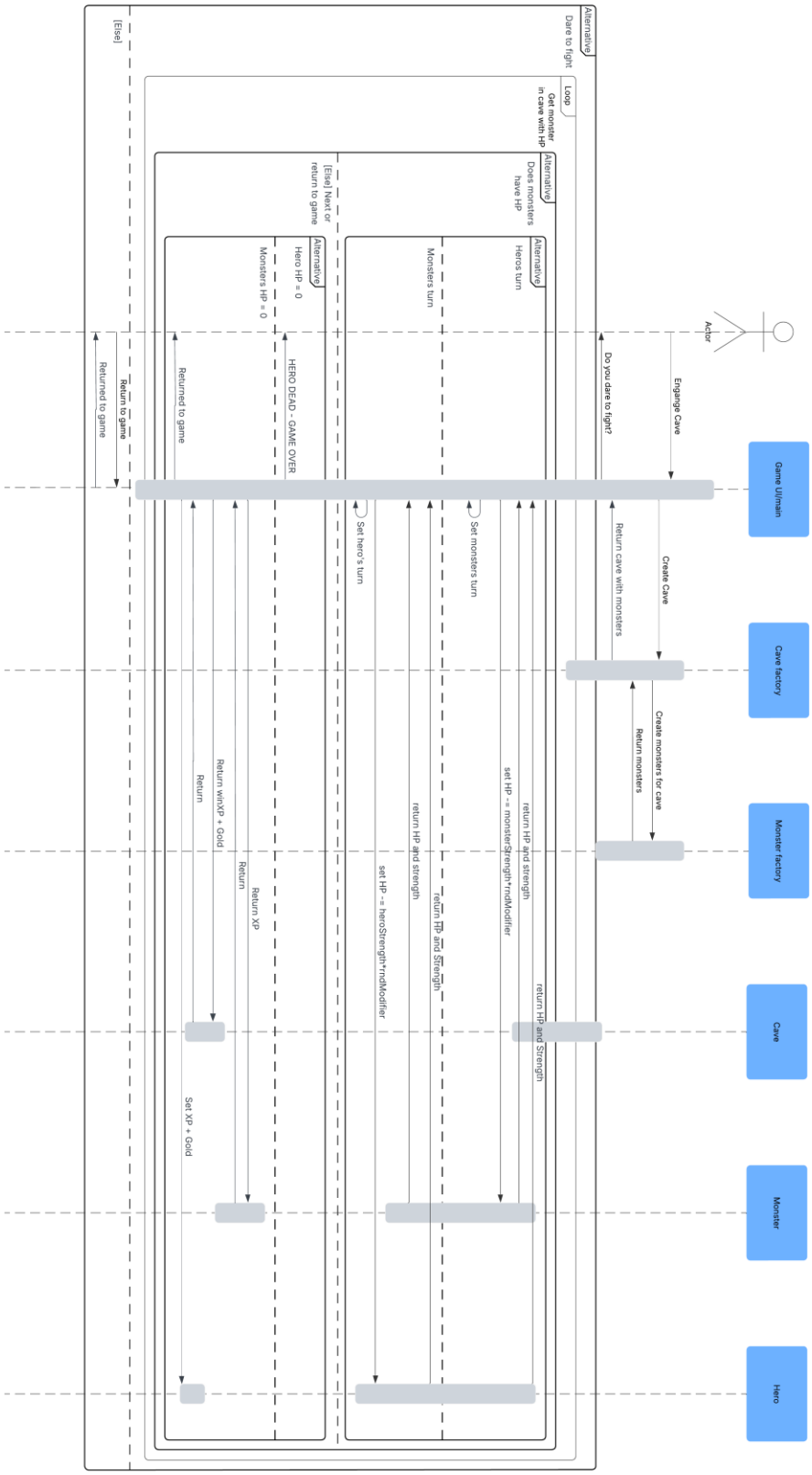


2. iteration

MonsterFight

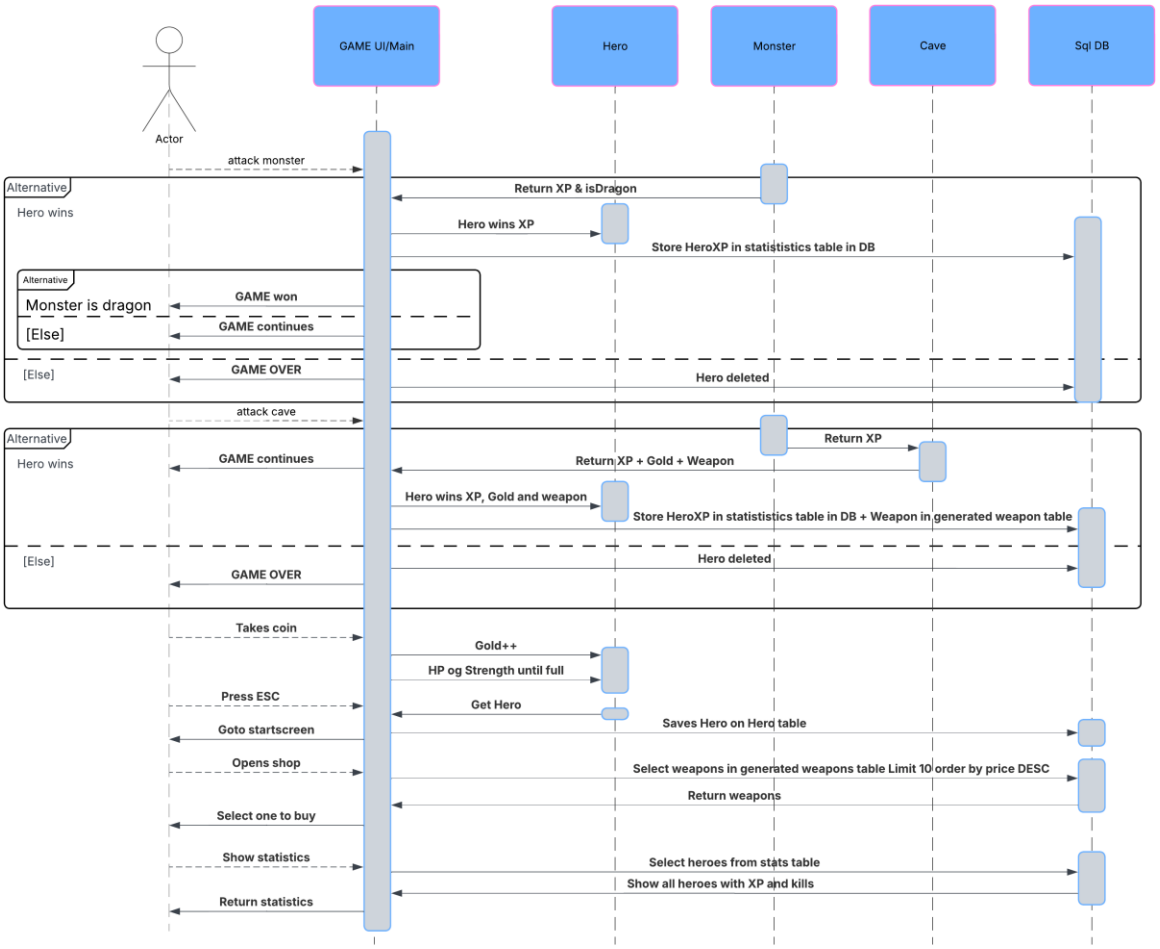


Cave fight

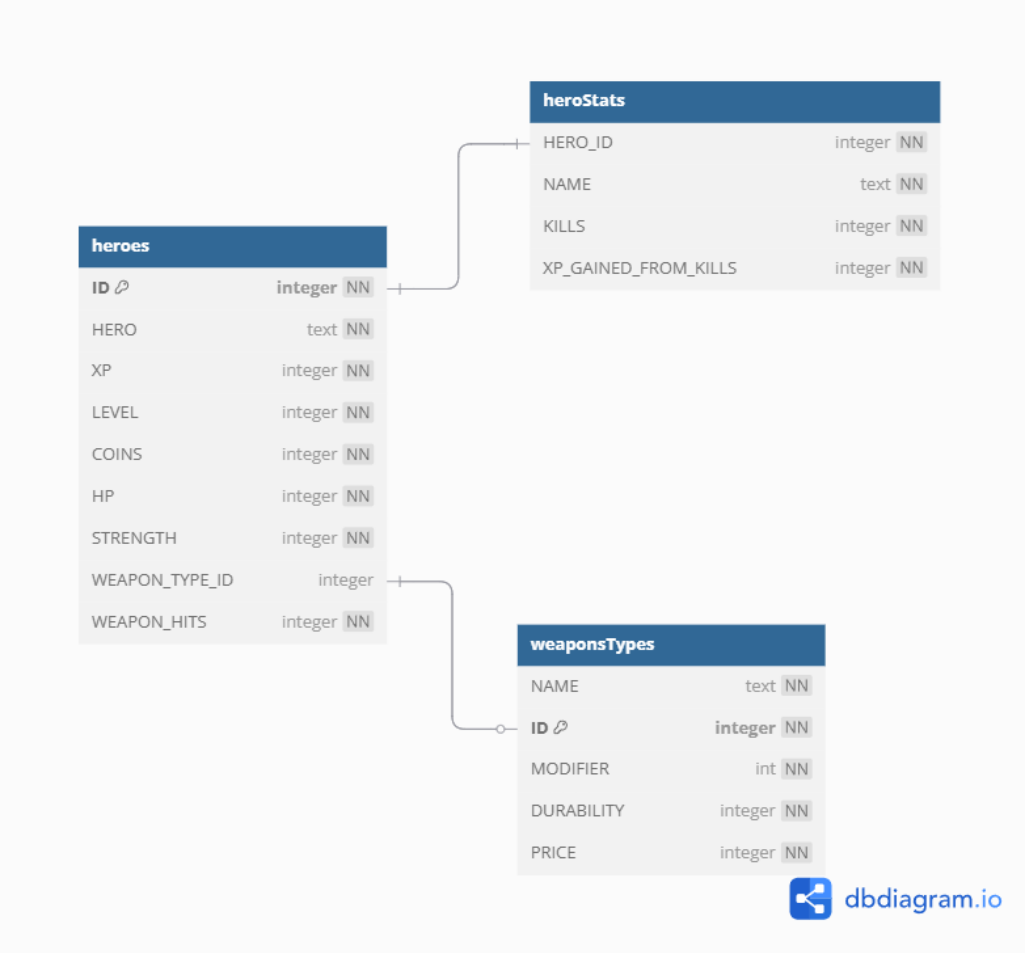

Actor
Game UI/main
Cave factory
Monster factory
Cave
Monster
Hero
Engange Cave
Do you dare to fight?
Create Cave
Return cave with monsters
Create monsters for cave
Return monsters
Alternative
Dare to fight
Loop
Get monster in cave with HP
Alternative
Does monsters have HP
Alternative
Heros turn
Set monsters turn
return HP and strength
set HP -= monsterStrength*rndModifier
Alternative
Monsters turn
Set hero's turn
return HP and strength
set HP -= heroStrength*rndModifier
[Else] Next or return to game
Alternative
Hero HP = 0
HERO DEAD - GAME OVER
Alternative
Monsters HP = 0
Returned to game
Return winXP + Gold
Return
Return XP
Return
Set XP + Gold
[Else]
Returned to game
Return to game
Returned to game
return HP and Strength
return HP and Strength

3. Iteration

End



SQL


heroStats
HERO_ID integer NN
NAME text NN
KILLS integer NN
XP_GAINED_FROM_KILLS integer NN
heroes
ID integer NN
HERO text NN
XP integer NN
LEVEL integer NN
COINS integer NN
HP integer NN
STRENGTH integer NN
WEAPON_TYPE_ID integer
WEAPON_HITS integer NN
weaponsTypes
NAME text NN
ID integer NN
MODIFIER int NN
DURABILITY integer NN
PRICE integer NN
dbdiagram.io