# HÁZI FELADAT

Programozás alapjai 2.

Végleges

György Márk Attila
ZCVPZT

2021. május 4.

# 1. Feladat

Kalandjáték

Tervezzen objektummodellt kalandjáték objektumainak leírására! Legyen pálya, harcos, szörny, és legyenek tárgyak, amit a harcos fel tud szedni. Határozza meg az objektumok kapcsolatát és felelősségét!
Demonstrálja a működést külön modulként fordított tesztprogrammal! A játék állását nem kell grafikusan megjeleníteni, elegendő csak karakteresen, a legegyszerűbb formában! A megoldáshoz **ne** használjon STL tárolót!

# 2. Feladatspecifikáció

A játék kiírja a standard kimenetre pályát szövegesen (pl:'#' a fal,'@' a player…) A felhasználó a „wasd" gombokkal tud majd mozogni. Fog tudni tárgyakat felszedni a földről, amiket tud majd használni, és ezek is láthatóak lesznek a pályán. Fog tudni A pályán lévő ellenségekkel harcolni. A játék célja, hogy eljusson a kijáratig.

# 3. Pontosított feladatspecifikáció

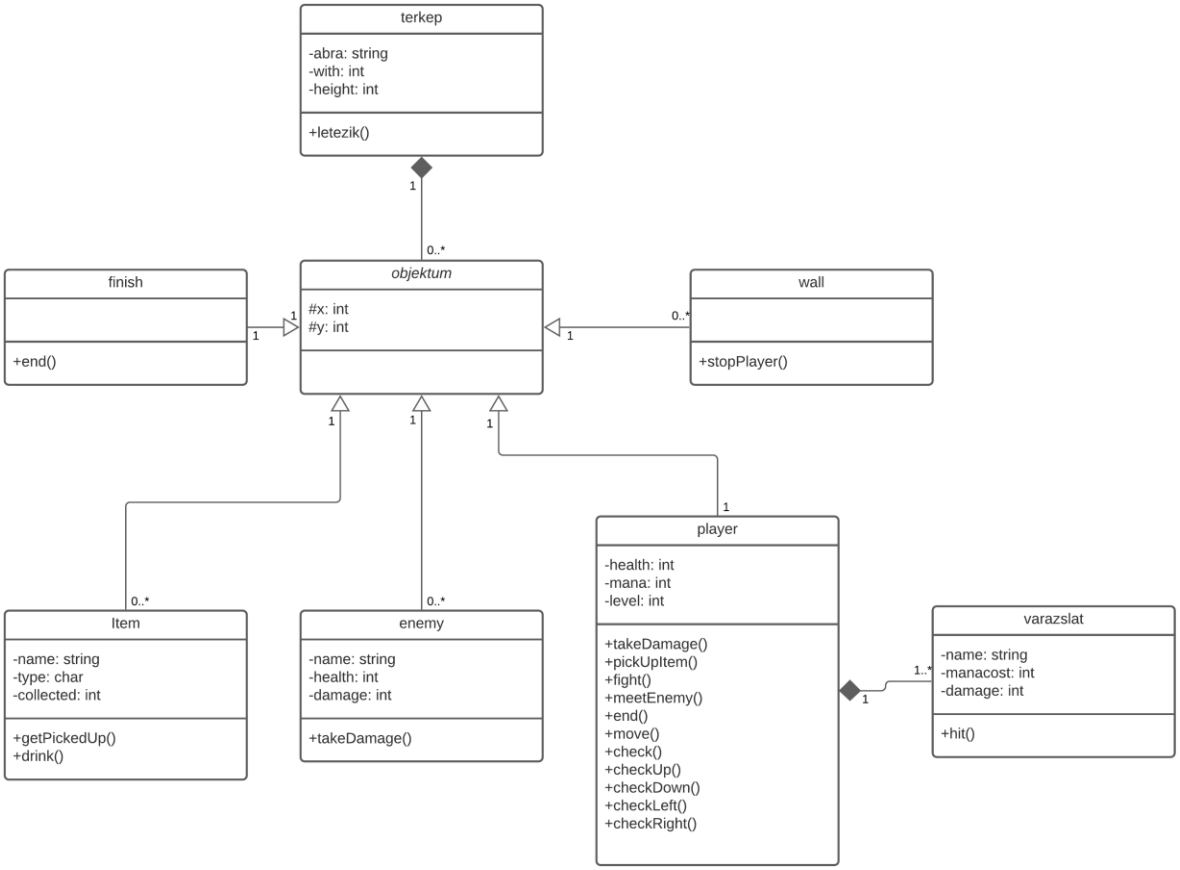
## 3.1. A bemenetek:

- Egy szövegesfájl, amiben benne van, hogy mekkora a pálya, és maga a pálya.
- Egy szövegesfájl, amiben el van tárolva az előző mentés: a játékos adatai, az eszköztára.
- A felhasználó „wasd" gombokkal tudja irányítani a karaktert.

## 3.2. A kimenetek:

- Minden lépés után frissül a pálya, mivel máshol van a játékos.
- Mindig ki vannak írva a játékos adatai, eszköztára.
- Mentésnél elmenti a játékos adatait, a pálya állását, az eszköztárát egy szövegesfájlba.

# 4. Terv
## 4.1. Osztálydiagram




**1**

## 4.2. Osztályok

### 4.2.1. Terkep:

- String abra: Ebben lesz a pálya eltárolva.[2]
- int with: A pálya szelessége.
- int height: A pálya magassága.
- letezik(): Létre lett-e hozva a pálya.

### 4.2.2. Objektum:

- int x: Az adott objektum x koordinátája.
- int y: Az adott objektum y koordinátája.

[1]Megváltozott az osztálydiagramm, a PDF végén csatolom az eredetit.
[2] char* volt eredetileg.

### 4.2.3. Wall:

- stopPlayer(): erre nem mehet a játékos

### 4.2.4. Finish:

- end(): Vége a játéknak

### 4.2.5. Item:[3]

- String name: Az adott tárgy neve.[4]
- char type: Az adott tárgy típusa.
- int collected: 0, ha a játékos még nem vette fel a tárgyat, 1, ha a játékos felvette, de nem használta, 2, ha a játékos használta a tárgyat.[5]
- getPickedUp(): Felveszi a játékos.
- drink(): Megissza a játékos.

### 4.2.6. Enemy:

- String name: Az ellenség neve.[6]
- int health: Az elenség élete.
- int damage: Az ellenség sebzése.
- takeDamage(): Sebződik az ellenség.

### 4.2.7. Player:

- int health: A játékos élete.
- int mana: A játékos mana szintje:
- int level: A játékos szintje (ez alapján nő a képességeinek a sebzése)
- takeDamage(): Sebződik a játékos.
- pickUpItem(): Felvesz egy tárgyat.
- fight(): Harcol az ellenséggel.
- meetEnemy(): A játékos egy olyan koordinátára lépett, ahol van ellenség.[7]
- end(): Vége van a játéknak.
- move(): Elmozdul
- CheckUp/Down/Left/Right(): Le ellenőrzi, hogy tud-e fel/le/balra/jobbra menni.
- check(): Le ellenőrzi, hogy tud e mozdulni az adott irányba az előző függvények segítségével.[8]

### 4.2.8. Varazslat:

- String name: A varázslat neve.
- int manacost: A varázslat ennyi manát használ el.
- int damage: A varázslat ennyit sebez az ellenségbe.
- hit(): Megsebzi az ellenséget

[3] Eredetileg Potion volt, de kicseréltem Item-re, így lehet hozzáadni többféle tárgyat is.
[4] char* volt eredetileg.
[5] Eredetileg nem volt benne, de kellett, hogy el lehessen dönteni, hogy a tárgy jelenleg hol helyezkedik el.
[6] char* volt eredetileg.
[7] Eredetileg nem volt benne, azért került bele, mert így egyszerűbb volt megkeresni, hogy a játékos olyan mezőre lépett-e, amin ellenség van, és ha igen akkor melyik az az ellenség.
[8] Eredetileg nem volt benne, nem feltétlen kellett, de így egyszerűbben néz ki szerintem.

# 5. Megvalósítás

A feladat megoldása 4-es pontban említett osztályok, és a főprogram megvalósítását igényelte. A főprogramban tesztelődnek le a függvények, ezért külön nincs hozzá tesztprogram. A programhoz használtam a laboron megcsinált String osztályt és a z előadáson vett lista sablonosztályt. A tervezési részben meghatározott osztályok megváltoztak. Továbbá a tervezési részben leírt két algoritmus is megváltozott, mert nem rak az abra-ba';' karaktereket, hanem a szélesség alapján ír ki endl karaktereket.

# 6. Tesztelés

## 6.1. Void mentesteszt()

Erre az egy tesztre volt szükség, mert az alap programban a mentés nem készül el a megadott inputtal, ezért van definiálva a teszt, és a főprogram végén létrejön a mentésfájl.

## 6.2. Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. Ehhez minden önálló fordítási egységben include-oltam a" memtrace.h" állományt a standard fejlécállományok után. Memóriakezelési hibát nem tapasztaltam a futtatások során.

## 6.3. Lefedettségi teszt

A főprogram a megadott inputtal lefeedte a program összes ágát.
A Jporta 4 olyan kódrészletet jelölt meg, ami nem futott a programban. Ezek a következők:

- A map.cpp-ben a letezik függvényben az a két ág, ami azt return-öli, hogy a térkép nem létezik.
- A main.cpp-ben a 3 befejezés közül kettő: az, hogy a játékosnak elfogy az életereje, és az, hogy a felhasználó kilép a játék közben, és csak ebben az esetben készülne mentés, ezért kellett a mentesteszt(), hogy a program azon része is lefusson.

[9]

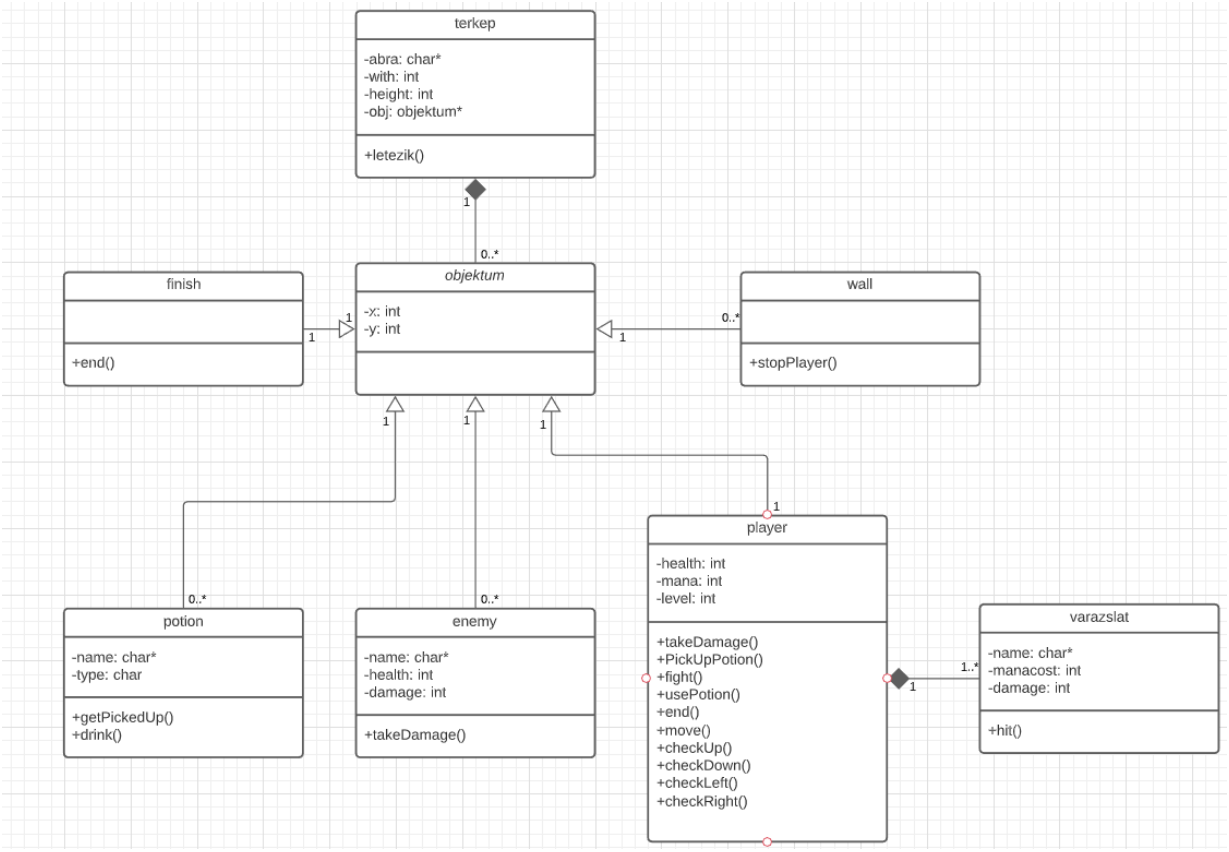
```
51. bool terkep::letezik(){
52.     ifstream f;
53.     f.open("map.txt",ios::in);
54.     if(!f.is_open())
55.         return false;
56.     char c;
57.     while(f>>c&&c!=';')
58.     f>>c;
59.     while(f>>c){
60.         if(c=='#'||c=='@'||c=='F'||c=='h'||c=='m'||c=='M'||c=='_')
61.             return false;
62.     }
63.     return true;
64. }
```

[9] map.cpp

```
144.    else if(plyr.end()){
145.        cout<<t;
146.        cout<<plyr;
147.        vonal();
148.        cout<<"You are dead!";
149.        ofstream s;
150.        s.open("save.txt",ios::out);
151.        s.close();
152.    }
153.    else
154.        mentes(t,plyr,en,i);
155.    file.close();
156.    g.close();
```

10

# 7. Eredeti osztálydiagram




[10] main.cpp