Házi Feladat

Programozás alapjai 2.

Takács András

1 Feladat:

A feladat egy 2 dimenziós játék, amiben kettő játékos versenyzik azon, hogy a másik játékos tetejére ugorjon ezzel pontot szerezzen. A játékosokat a billentyűzetről lehet irányítani és egy grafikus ablakban jelenik meg. A játék rendelkezik előre elkészített pályákkal, de a felhasználó is készíthet extra pályákat.

2 Feladatspecifikáció:

A feladat egy két fős játék elkészítése, melyben a két játékos egy-egy színes négyzetet irányít és a játék célja az, hogy a játékos a másik játékos által irányított négyzet tetejére ugorjon, ezzel pontot szerezve.

Amikor egy játékos pontot szerezt azt a játék egy képernyő közepére, a játékos színével megegyező, kiírt szöveggel jelzi, kiírja a játékos pontjainak a számát és egy rövid idő után visszaállítja a két játékost az eredeti helyükre. Amennyiben egyik játékos eléri a pályához tartozó győzelemhez szükséges pontszámot a program ezt szintén kiírással jelzi és egy új pályát tölt be nullázott pontokkal.

A két játékos egyazon számítógépről játszik, egy billentyűzeten osztozkodva, a kék játékos az A és D billentyűket használhatja horizontális mozgáshoz, a space-t ugráshoz és az S-et egy azonnali lefele gyorsuláshoz, míg a piros játékos esetében ezek megfelelője a két oldal nyíl a bal shift és a lefele nyíl.

A pálya, a játékosok és a szövegek grafikusan az SDL3 könyvtár segítségével vannak megjelenítve és a billentyűzet kezelésében is ez a könyvtár fog segíteni. A két játékos fizika szimulációjához a kód számontartja a gyorsulásukat és sebességüket, emellett biztosítja a helyes ütközéseket a másik játékossal és a pályaelemekkel. Ha egy játékos kimegy a pálya oldalán akkor a pálya másik oldalán bukkan fel, ha kiesik a pálya alján akkor a pálya tetejére kerül. Mivel a két játékos és a pályák is téglalapok, ezért az ütközések detektáláshoz elég azt ellenőrizni, hogy tetszőleges két tengelyhez képest el nem forgatott téglalap metszi-e egymást. Egyes pályaelemekről az ütközéskor visszapattannak a játékosok, más pályaelemek megölik a játékost.

A program a grafikus megjelenítés mellett egy parancssorban rögzíti a fontosabb eseményeket/fellépő hibákat. Egy ilyen lehetséges hiba például egy hibás pályafájl, ebben az esetben a program a terminálban jelzi, hogy melyik fájl, mely sorában keresendő a hiba. A pályákat „.gamemap” kiterjesztésű fájlokból az induláskor tölti be a program, minden a játékkal egy mappában lévő ilyen fájlt betöltésével megpróbálkozik. A formátum pontosabb specifikációja majd a részletesebb tervben lesz megtalálható.

3 Pontosított specifikáció:

A pályafájlok formátuma az alábbiaknak fog megfelelni:

*Pálya magassága(double, a tényleges magasság ennek a kétszeres)*

*Pálya háttérszíne(3 x byte)*

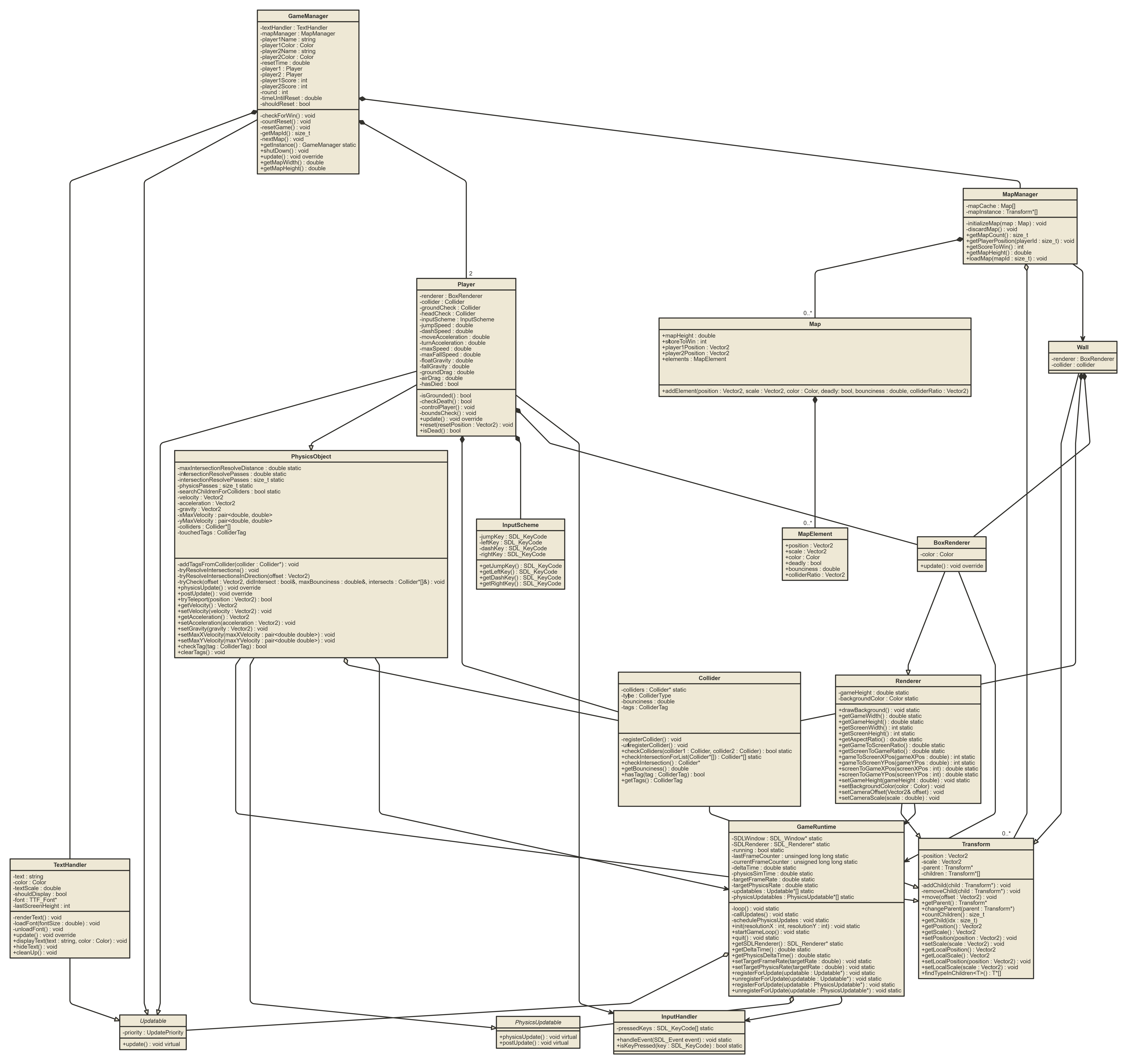
*Első játékos kezdőkoordinátái(float float)*

*Második játékos kezdőkoordinátái(float float)*

*N db falelem:(pozíció: float float, méret: float float, szín: 3 x byte, visszapattanás mértéke: float, fizikai kiterjedés aránya: float, float)*

4 Terv:

Az UML diagramot a <https://www.nomnoml.com> segítségével hoztam létre, részletezi a projekthez tartozó osztályokat és a közöttük lévő kapcsolatokat.



(Bele kell zoomolni hogy olvasható legyen)

5 Fontosabb algoritmusok:

5.1 Kettő el nem forgatott téglalap metszi-e egymást

Ezt az algoritmust használom arra, hogy ellenőrizzem, hogy két játékbeli objektum ütközik-e.

Tudjuk: center1, oldalhosszak1, center2, oldalhosszak2

//ha az első téglalap alja magasabban van a második téglalap tetejénél, így biztos nem metszgetik egymást

IF center1.y - oldalhosszak1.y/2 > center2.y + oldalhosszak2.y/2:

RETURN HAMIS

//ha az első téglalap teteje alacsonyabban van a második téglalap aljánál, így biztos nem metszhetik egymást

IF center1.y + oldalhosszak1.y/2 <center2.y – oldalhosszak2.y/2:

RETURN HAMIS

//ha az első téglalap bal széle jobbra van a második téglalap jobb oldalánál, így biztos nem metszhetik egymást

IF center1.x - oldalhosszak1.x/2 > center2.x + oldalhosszak2.x/2:

RETURN HAMIS

//ha az első téglalap jobb széle balra van a második téglalap bal oldalánál, így biztos nem metszhetik egymást

IF center1.x + oldalhosszak1.x/2 <center2.x - oldalhosszak2.x/2:

RETURN HAMIS

//különben kizárásos alapon a kettő téglalap metszi egymást

RETURN IGAZ

5.2 Fizikai objektum szimulációja:

A fizikai objektum rendelkezik sebesség- és gyorsulásvektorral, minden szimulációs lépés elején a sebességhez hozzáadja a gyorsulást szorozva a szimulációk gyakoriságának reciprokával, hogy konzisztens viselkedést biztosítson a képkockák számától függetlenül. Mivel a gyorsulás négyzetes kapcsolatban van a megtett úttal a szimulációs rátát konzisztensen tartjuk, hogy az ebből adódó hibákat elkerüljük.

Kiszámoljuk a lehetséges mozgatás maximumát: ez sebességvektor szorozva a szimulációk gyakoriságának reciprokával.

Felvesszük a próbamozgatás nagyságát: először ½, aztán 1/4 … ½^n

Először a x tengely mentén mozgatjuk a próbamozgatás x komponensével, ha nem ütközik semmi mással maradhat, ha nem visszaléptetjük a jelenlegi próbamozgatással

Ezt az y tengely mentén is megtesszük

Ezután felezzük a próbamozgatást, amíg el nem érünk ½^n-ig

Amennyiben a feljebb lévő algoritmus során ütköztünk valamivel, nullázzuk a sebességet és gyorsulást, amennyiben volt az ütközések között visszapattanós tulajdonságú objektum, akkor ennek arányában megfordítjuk a sebességet.

Az algoritmus garantálja, hogy az objektumok relatíve jól hozzásimulnak egymáshoz, n számától a lebegőpontos számítás hibáitól függően.

5.3 Két fizikai objektum metszésének a feloldása:

Bizonyos esetekben (leggyakrabban egy pálya betöltésekor) előfordulhat, hogy két játékos vagy egy játékos és egy pályaelem, ugyanazt a pozíciót veszik fel, a fenti algoritmus segítségével az alábbi helyzet csak akkor oldódhatna fel ha a játékos(ok) egy képkockán belül tudnak annyit mozogni, hogy a helyzet megszűnjön. Ez magas precizitással csak ritkán fordul elő, maximum csak akkor a két objektum nincs nagyon mélyen egymásba ágyazódva.

Az ilyen helyzetek feloldásának érdekében minden fizikai objektum a szimulációs lépése elején ellenőrzi, hogy a fenti eset fenn áll-e és amennyiben fennáll az alábbi módon kezeli:

Minden kardináls(fel, le, jobbra, balra): Megpróbálja végrehajtani az álábbi algoritmus, amennyiben az adott irány sikeres kilép, ha nem visszállítja és a következő iránnyal is megpróbálja:

Az objektumot elmozgatja a maximum feloldási távolsággal az adott irányba, amennyiben a metszés megszűnt sikeresnek találja, ha nem nem. Az objektumot visszamozgatja az előző távolság felével és ellenőrzi, hogy így már jó-e, ha jó eltárolja a távolságot és feljegyzi, hogy jó, ha nem nem. Ezt a bináris keresés jellegű szűkítést megteszi n db alkalommal és folyamatosan számontartja, hogy volt-e olyan távolság, ami fel tudja oldani a metszést és hogy mi volt a legkisebb ilyen távolság.

Megjegyzés: A fentebbi algoritmus célja főként az, hogy hibásan megtervezett pályákon (ahol a játékos beragadna a falba) se haljon meg a játék és hogy a váratlan kisebb mozgási hibákat kezelni tudja. Az algoritmus, sok triviális szemmel megoldható esetet nem fed le, viszont segíteni tud a tesztelés során megtapasztalt valós helyzetekkel.

6 Pálya fájlformátum specifikáció:

A pályafájlokat a játék az elinduláskor beolvassa, a benne talált értékeket Map adatstruktúrákban tárolja melyek, pályaspecifikus adatokat tartalmaznak; pl.: háttérszín, pálya magassága, míg az egyes elemek adatait MapElement adatstruktúrákban tárolja. A pályákat egy MapCache nevű Map-ekből álló listában tárolja és pályaváltáskor a kiválasztott pályát ez alapján inicializálja.

A Map adatstruktúra felépítése:

{

Pálya magassága: Lebegőpontos

Háttérszín: Szín

Nyeréshez szükséges pontszám: Egész

1. Játékos kezdőpozíciója: Lebegőpontos vektor
2. Játékos kezdőpozíciója: Lebegőpontos vektor

Pályaelemek: MapElementek listája

}

A MapElement adatstruktúra felépítése

{

Pozíció: Lebegőpontos vektor

Méret: Lebegőpontos vektor

Szín: Szín

Halálosság: Logikai

Visszapattanási együttható: Lebegőpontos

Collider arány: Lebegőpontos vektor(skalárokat tartalmaz)

}

Egy .gamemap fájl felépítése:

(Az elemek whitespacekkel vannak szeparálva, a konkrét fájlokban sortöréseket is használok, de ezt csak az olvashatóság kedvéért, a sortörés nélküli fájlok is validnak számítanak a program által, habár személy szerint én ellenzem őket.)

#INT/PÁLYAMAGASSÁG#

#INT/HÁTTÉRSZÍN\_R# #INT/HÁTTÉRSZÍN\_G# #INT/HÁTTÉRSZÍN\_B#

#INT/NYERÉSI\_PONTSZÁM#

#FLOAT/JÁTÉKOS\_1\_POZ\_X# #FLOAT/JÁTÉKOS\_1\_POZ\_Y#

#FLOAT/JÁTÉKOS\_2\_POZ\_X# #FLOAT/JÁTÉKOS\_2\_POZ\_Y#

(itt következik tetszőleges mennyiségű pályaelem)

#FLOAT/ELEM\_N\_POZ\_X# #FLOAT/ELEM\_N\_POZ\_Y# #FLOAT/ELEM\_N\_MÉRET\_X# #FLOAT/ELEM\_N\_MÉRET\_Y# #INT/ELEM\_N\_SZÍN\_R# #INT/ELEM\_N\_SZÍN\_G# #INT/ELEM\_N\_SZÍN\_B# #BOOL/HALÁLOS\_E# #FLOAT/VISSZA\_PATTANÁS# #FLOAT/COLLIDER\_X# #FLOAT/COLLIDER\_Y#

(minden pályaelem értelemszerűen egy külön MapElement-be kerül, amiknek a listája a Map-hez tartozik)

7 Fordítás/futtatás:

A projektet g++-al fordítottam, CPORTA flaggel nem igényli az SDL3 és az egyéb szükséges fájlok jelenlétét, ekkor csak a két tesztpályafájl kell(más pályák nem is lehetnek, mert akkor nem fut le a teszt). Ekkor csak a tesztek és a teszteléshez szükséges részek fordulnak le. A játék így nem lesz játszható.

Ha CPORTA flag nélkül fordítunk, akkor egy játszható játékos kapunk, de ehhez szükséges:

* SDL3 és SDL3\_ttf DLL-ek rendelkezésre állása és az -lSDL3 -lSDL3\_ttf flagekkel való fordítás
* Játszható pályák a mappában(ezek lehetnek tesztpályák is)
* Egy „gamefont.ttf” betűtípus szintén a mappában legyen, hogy legyen mivel megjeleníteni a játékban lévő szövegeket

8 Megvalósítás:

A program megvalósítása nem különbözik jelentősen a fenti diagramban megadottól, a fő változás annyi, hogy bekerül egy teszt.cpp és egy teszt.h is, ezek a teszteléssel segítenek, valamint a CPORTA flag a program egyes részeit letiltja/engedélyezi.

**A CPORTA-s verzióban dirent.h-t használok filesystem helyett mivel a filesystem-et a JPORTA nem fogadja el, annak ellenére, hogy elvileg CPP17-el fordít.**

9 Tesztelés:

A program a tesztjei a kb 1500 soros teszt.cpp file-ban találhatóak. A tesztek kiterjednek a:

* Vector2 adatstruktúra megvalósítására
* A Transform osztályra és ennek helyes viselkedésére (megfelelő lineáris transzformációk)
* A Collider osztályra és ennek helyes viselkedésére (megfelelő metszésdetektálás)
* A PhysicsObject osztályra és ennek helyes viselkedésére(egy módosított GameHandler segítségével konkrét esetek helyes szimulációja)
* A pályák betöltésére és hogy ezek megegyeznek a tesztesetben definiált értékekkel

10 Dokumentáció: