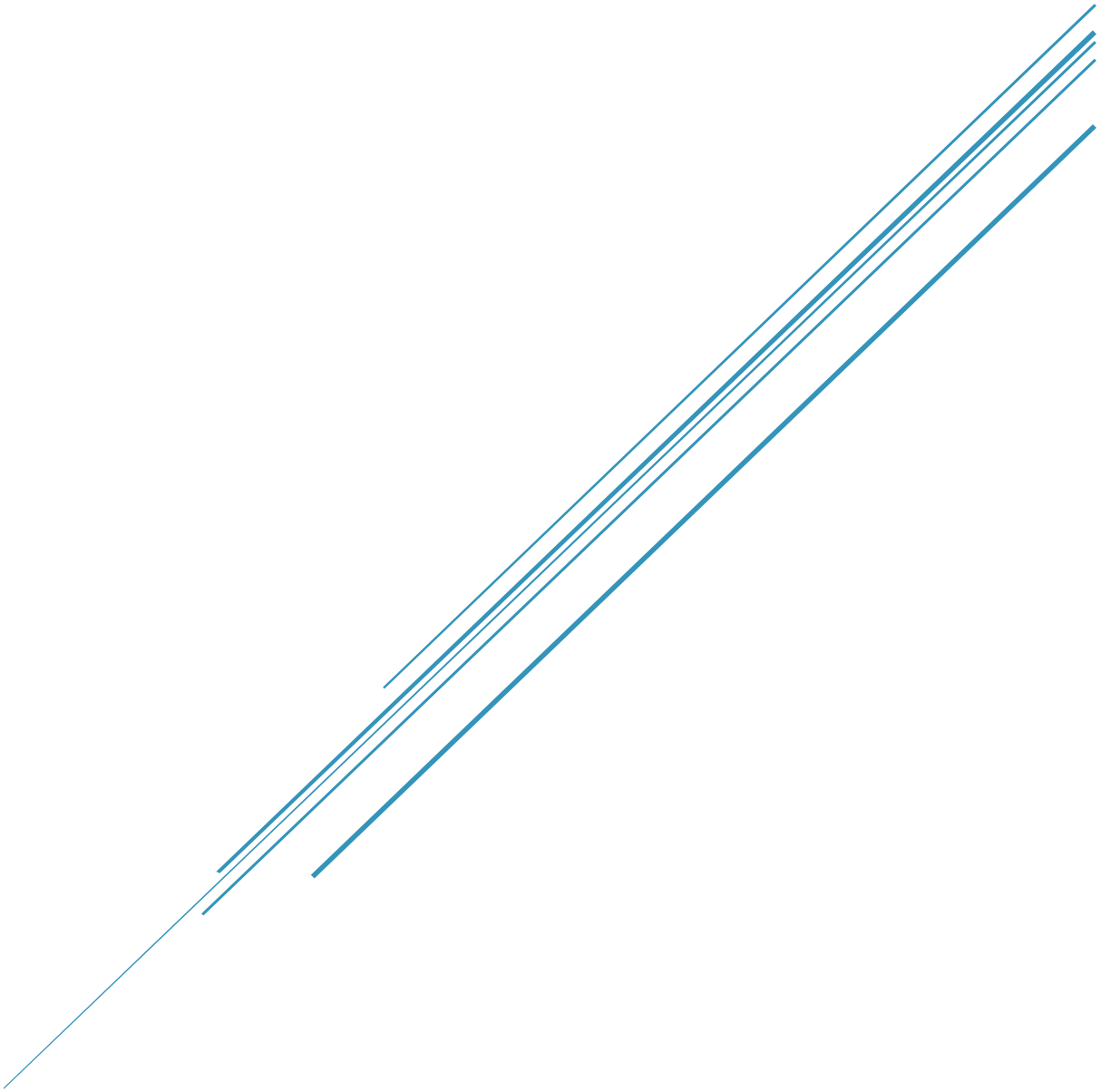
Töltés játék

Programozói dokumentáció



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Programozás alapjai I.

Tartalom

Bevezető, a játék alapjai

A játékban a játékos célja az, hogy a pályán található, valamilyen kezdősebességgel rendelkező protont eljuttassa a célponthoz. A proton egy pozitív töltésű részecske. Más, a pályán található töltések közül a pozitív töltések taszítják, a negatív töltések pedig vonzzák a protont. A játékos ezek segítségével irányíthatja a protont a célhoz.

A játék pályáin lehetnek falak, amelyekbe a proton beleütközik, és fix töltések, amelyeket a játékos nem tud módosítani. A játékos feladat az, hogy valamilyen konfigurációban töltéseket helyezzen el a pályán, ezután útjára indíthatja a töltést. Ha töltés nem ér célba, a játékos alaphelyzetbe állíthatja a protont, módosíthat a lerakott töltésein, és újra próbálkozhat. A játékos a pálya közepén is megállíthatja a protont, hogy módosítsa a lerakott töltéseit, de ekkor több pontot veszít, mint az alaphelyzetbe állításkor. A proton mozgása közben a játékos nem változtathatja meg a pályát.

A töltéseket a játéktéren látható rács pontjaiba lehet elhelyezni. Ezt a bal, illetve jobb egérgombokkal teheti meg a játékos. Amennyiben a rács adott pontjában még nincs töltés, a kattintások létrehoznak egy (pozitív, illetve negatív) töltést, egyébként a bal egérgomb a felhasználó által elhelyezett töltést növeli, a jobb egérgomb pedig csökkenti.

A játék rendelkezik egy statikus, a játékablak bal szélén elhelyezkedő menüvel, a játékos ennek használatával választhat a pályák közül, illetve irányíthatja próbálkozásait egy adott pályán.

Az öt pálya leküzdése után a játékos neve bekérésre kerül, és ha elég magas pontszámot ért el, felkerül a dicsőséglistára (a dokumentációban ez többször toplista néven lesz megemlítve). Ezt a listát szintén meg lehet tekinteni a megfelelő menüpont kiválasztásával.

Megjegyzés: A játék egy log fájlt vezet (stdout.txt), amelyben nyomon követhetőek a működésének lépései. Ebbe a fájlba írnak a kódban található printf() utasítások.

Adatszerkezetek

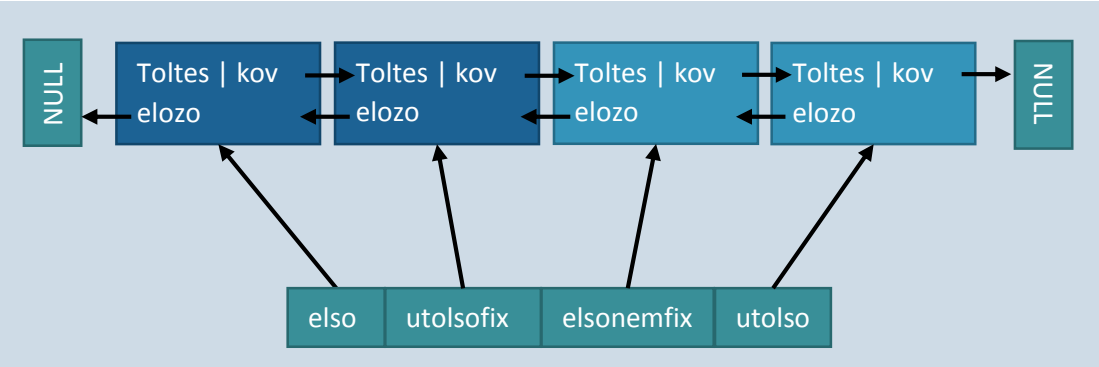
A programban dinamikus adatszerkezetek használatára többször is szükség van, ezek fajtái és működése itt látható.

Rácspontok

A rácspontokat tároló kétdimenziós tömb méretei az ablakmérettől függnek, ezért ezt egy egyszerű dinamikus tömbben tárolhatjuk, mivel futásidőben már kiderülnek a játékablak paraméterei (a pályákat tartalmazó fájl beolvasásakor).

Töltések

Mivel számuk a játék futása közben tetszőleges módon változhat, a töltések egy speciális, duplán láncolt listában vannak tárolva. A lista különlegességét az adja, hogy fix és nem fix töltéseket egyaránt tartalmaz, és ezeket néha külön is kell kezelnünk. A lista kezeléséhez használt pointerek az ábrán, függvények pedig a toltesek.c fájl részleteinél láthatóak.



A töltések láncolt listája

Az ábrán nem szereplő töltés kezelő pointerek:

```
ToltesLista *uj;
ToltesLista *mozgoptr;
```

A töltések listaelem struktúrája (toltesek.h):

```
typedef struct tolteslista {
    Toltes toltes;
    struct tolteslista *kov;
    struct tolteslista *elozo;
} ToltesLista;
```

Maga a töltés egy külön struktúra lesz, mivel azt később könnyebb lesz átadni függvényeknek (például kirajzolásnál).

Falak

A falak láncolt listája ennél sokkal egyszerűbb. Ez a lista nem duplán láncolt, mivel bejárására csak egy irányban van szükség. Az egyszerűség kedvéért eltároljuk az utolsó elemre mutató pointert is, így nem kell végigmenni a listán, amikor egy új falat szúrunk be (így a listában a falak sorrendje megegyezik a pályát tartalmazó fájlban lévő falak sorrendjével, csak ezért nem az elejére szúrjuk be az új falakat).

A falakat kezelő pointerek:

```
FalLista *uj_fal;
FalLista *mozgoptr_fal;
FalLista *elso_fal;
FalLista *utolso_fal;
```

Láncolt listák felszabadítása

Minden pálya betöltésekor szükség van a két láncolt lista teljes felszabadítására, hogy oda az új pálya elemei kerülhessenek. Ezt a clearlevel() függvény végzi el.

```
mozgoptr_fal = elso_fal;
FalLista *torol_fal;
while (mozgoptr_fal != NULL) {
    torol_fal = mozgoptr_fal;
    mozgoptr_fal = mozgoptr_fal->kov;
    free(torol_fal);
}
elso_fal = utolso_fal = NULL;
```

A függvény fenti részlete a falak listájának felszabadítását végzi, a töltések listájának törlése ugyanilyen módon történik. A lista bejárása közben egy ideiglenes pointerbe mentésre kerül a mozgó pointer aktuális pozíciója, majd a mozgó töltés továbbhaladása után felszabadításra kerül a memóriaterület. A bejárás végén a listát kezelő pointerek az új és a mozgó pointer kivételével NULL értéket kapnak.

Globális pointerek

A láncolt listákat kezelő pointerek az egyszerűség kedvéért mind globális változók, így nem kell minden őket használó függvénynek átadni őket. Nem lenne lényegesen különböző a pointereket a függvényeknek paraméterként átadó megoldás sem, de így átláthatóbbak maradtak a kódnak ezek a részei.

A program működését vezérlő fő függvények (main.c)

Általános függvények

void fatal_error (int errorcode)

Ha a program futása közben végzetes hiba történik (például hiányzó fájl, ami miatt nincs értelme a játék futásának, vagy memóriakezelési hiba), biztonságosan leállítja a program működését. A leállító függvény meghívásakor paraméterként a kívánt kilépési kódot kell megadni, például:

```
fatal_error(0xB4);
```

A konzolablakban az alábbi hibakódok jelenhetnek meg. A memóriahibák „A”, a fájlkezelési hibák „F” jelölést kaptak (Allocation error, File error).

Hibakód	Hiba leírása
0x01	Az SDL felület megnyitása közben történt hiba
0xA1	Játékos által töltés létrehozása közben nem sikerült memóriát lefoglalni
0xA2	Pálya betöltésekor fix töltés létrehozása közben nem sikerült memóriát lefoglalni
0xA3	Nem sikerült memóriát foglalni a rácspontokat tartalmazó tömb sorainak
0xA4	Nem sikerült memóriát foglalni a rácspontokat tartalmazó tömb oszlopainak
0xF1	Nem sikerült a dicsőséglista fájl megnyitása olvasásra
0xF2	Nem sikerült a dicsőséglista fájl megnyitása írásra
0xF3	Nem sikerült megnyitni a pályacsomagot tartalmazó fájlt
0xF4	Hiba történt egy pálya fájlból beolvasása közben (hibás levels.txt fájl)

```
Process returned 241 (0xF1)   execution time : 0.016 s
Press any key to continue.
```

Egy példa a program leállására és a hibakód kiírására
(a hiba a dicsőséglista fájl megnyitásakor történt)

A program sikeres futásakor és szándékos leállításakor az SDL felületből kilépés után a konzolablakban a (0x0) kód jelenik meg.

void clearlevel (void)

Felszabadítja a programban található két láncolt listát. Működését lásd itt.

void freememory (void)

Felszabadítja a rácspontokat tartalmazó dinamikus tömböt, majd meghívja a clearlevel() függvényt, így összességében felszabadítva minden, a játék által dinamikusan foglalt memóriát. A rácspontok felszabadításához két globális pointert használ, amelyek a sorok számának meghatározásának és a tömb foglalása után lesznek beállítva (ezek függnek a játékablak méretétől).

void closeSDL (void)

Leállítja az időzítőt, majd bezárja az SDL felületet. Az időzítőre egy globális pointer mutat.

int racspont_e (int x, int y, Pont **racspontmx, int sorok, int oszlopok, int *racsindexi, int *racsindexj, int toltesr)

Megnézi, hogy a kattintás koordinátái közel voltak-e egy rácsponthoz. Ott detektálja a kattintást, ahol a kattintás a rácspont koordinátáitól kisebb távolságra van, mint egy kirajzolt töltés sugara, ekkor igaz értéket ad vissza a függvény. A két cím szerint átadott integerbe írja annak a rácspontnak az indexeit, amelyiken a kattintás történt, és ezekkel történik utána a töltéseket módosító függvények hívása.

Célpont kezelése

int celpont_hit(Mozgotoltes proton, int celpontr)

Igaz értéket ad vissza, ha a proton hozzáér a töltéshez. Bemenetként szüksége van a proton struktúrájára, illetve a célpont sugarára.

void celpont_rajzol (SDL_Surface *screen, int celpontr)

Kirajzolja a célpontot a képernyőre.

Dicsőséglista

A játékban legtöbb pontot elért játékosok listáját a program fájlban tárolja. A játék megnyitásakor ez a fájl beolvasásra kerül, a futás közben a memóriában van kezelve a lista, majd a játék után újra a fájlba lesz írva, felülírva az előző, ott található listát.

Futás közben egy struktúrákból álló tömbben van a lista, ez a tömb hsdb + 1 elemű. A nulladik elembe történik a játékos által elért pontszám beolvasása, a többi elemben pedig sorba rendezve van a többi pontszám (legjobb az első, legrosszabb a hsdb indexű elem). A hsdb alapértelmezett értéke 10, ez lesz a kiírásra kerülő rekordok száma.

A pontszámok struktúrája egy integert és egy maximum 16 karakter hosszú nevet tartalmaz.

```
typedef struct {
    int score;
    char name[16+1];
} highscore_entry;
```

void highscores_read (highscore_entry *highscore_list, int hsdb)

Megnyitja a highscores.txt fájlt, majd beolvassa a toplistát. A toplista fájl sorai pontszám;név formátumúak. Ezeket a rekordokat a highscore_list[] tömbbe írja bele.

void highscores_write (highscore_entry *highscore_list, int hsdb)

Kiírja a tömbben tárolt toplista adatokat a highscores.txt fájlba, felülírva az előző adatokat.

void highscores_sort (highscore_entry *highscore_list, int hsdb)

Sorba rendezi a toplista elemeit. Ehhez először a legkisebb pontszámú elemet a tömb nulladik helyére rakja, a második részben pedig maximum kiválasztással rendezi a többi elemet. Ez az algoritmus megfelel a célnak, mivel a lista rendezésekor a futásidő nem szempont.

Pályák

void loadLevel (int lvl, Mozgotoltes *proton)

Beolvassa egy betöltendő pálya adatait. Paraméterként a pálya számát, illetve a protont kapja meg, utóbbit cím szerint, mivel módosításra kerül a proton pozíciója és sebessége.

A pályákat tartalmazó fájlt soronként olvassa be, annak az alábbi formátumúnak kell lennie:

```
B3
250,400:cp
950:px
550:py
-0.5:vx
0:vy
F
F,1,450,200,350
T
T,300, 250, +1
T,550, 350, -2
E
```

A beolvasó függvény a keresett pálya kezdősorát (Bx) keresi meg, utána beolvassa az 5 sornyi adatot, amelyet minden pályának tartalmaznia kell (célpont koordinátái, a proton x és y koordinátái, illetve sebessége, komponensenként). Ezután, ha lesz fal a pályán, egy F karaktert tartalmazó sornak kell következnie, utána pedig F kezdőbetű után a falak adatainak, új sorokban. Ugyanilyen módon adhatóak meg a pálya töltései. A pálya leírásnak végét egy E betű jelzi.

A kép frissítése

A játék aktuális állapotától függően három különböző függvény végzi a képernyő újrarajzolását.

void frissit (SDL_Surface *screen, int beosztas, int elsooszlop, int toltesr, Mozgotoltes proton, int rstcounter, int pausecounter, Button *menu, int bt_shownumbers, int celpontr)

A játék normális működése közben futó függvény. Láthatóan nagyon sok változóra szüksége van a működéshez. Tudnia kell a rács méreteit, a proton helyét, a számlálók aktuális értékeit, és természetesen meg kell kapnia a menü tömbjét is, hogy meghívhassa a drawmenu() függvényt. A bt_shownumbers gomb indexére azért van szüksége, mert annak állapotától függően írja ki a töltések felé azok numerikus értékét. A falak és töltések kirajzolásához az azokat kezelő globális pointereket használja.

Működésének lépései a kódban vannak kommentelve. Röviden: kirajzolja a játékteret, töltésekkel, falakkal, protonnal, illetve a menüt.

void frissit_levelend (SDL_Surface *screen, int *start, int rstcounter, int pausecounter, int current, Button *menu, int bt_level1, int *highscoreclick, int buttonwidth, highscore_entry *highscore_list)

Ez a függvény veszi át a rajzolást, amikor véget ér egy pálya. Kiírja a játékos által a pályán elért pontszámot. Ha a játékos az utolsó (ötödik) pályát teljesítette, megjelenik a gomb, amivel a dicsőséglistához léphet, és beírhatja a nevét.

Pontszámok

Szintén ebben a függvényben számolódik ki a játékos pályán elért pontszáma, az alábbi módon:

```
(int)fmax(5000-pausecounter*750-rstcounter*500, 0);
```

A játékos 5000 pontról kezdi a pályákat, ebből a proton megállításával 750, alaphelyzetbe állításával pedig 500 pontot veszít. A pontszám nem mehet 0 alá, ezt szolgálja az fmax() függvény hívása.

Az öt pálya különböző adatai (köztük a pontszám is) az alábbi struktúrában vannak tárolva:

```
typedef struct {
    int score;
    int pausecounter;
    int resetcounter;
} Palyaadatok;
```

A programban egy ilyen struktúrákból álló, 6 elemű tömb van. A nulla indexű elem csak az összesített pontszám számolására van használva.

void frissit_hs (SDL_Surface *screen, highscore_entry *highscore_list, int hsdb, int textinput, int *endoftextinput, Button *menu)

Ez a függvény rajzolja a képernyőre a dicsőséglistát, illetve a szövegbeviteli mezőt, ha a játékos az ötödik pálya teljesítése után került erre a képernyőre. Ha a játékos befejezte a nevének bevitelét, és Enter gombot nyomott, meghívja a pontszámokat rendező függvényt.

A main függvény (main.c)

int main (int argc, char *argv[])

A main függvény a program leghosszabb függvénye, nagyrészt azért, mert itt történik az inputok kezelése.

A függvény változók létrehozásával kezdődik, például a proton és az eredővektor kerül itt létrehozásra. A változók magyarázatát lásd a kód kommentjeiben.

Megjegyzés: A színsémák számozva vannak, az itt létrehozott colorschemeid nevű változó tárolja az aktuális színséma számát. Ez a számozás nem része a színséma struktúráknak, hanem egy main függvényben található switch átírásával változtathatóak. Ez a switch a kódban a

```
/** Tema ujratoltes */
```

komment alatt található meg.

Ezután a rács létrehozása történik meg. A rács sűrűségét a beosztas nevű változó szabályozza. Ez tetszőleges, a töltések sugaránál nagyobb egész szám lehet (mivel egyébként összeérnének a töltések).

Rácspontok

A játékban töltések csak a játéktéren lévő rács pontjaiban helyezkedhetnek el. Minden ilyen pont (x, y) koordinátája tárolásra kerül egy racspontmx nevű kétdimenziós, dinamikus tömbben, Pont struktúraként. A program először a játékablak méretének adataiból meghatározza a szükséges sorok és oszlopok számát, majd lefoglalja a sorok*oszlopok darab pont tárolásához szükséges memóriát, végül feltölti a tömböt a megfelelő koordinátájú pontokkal.

A rácspontokra való kattintások kezelése a racspont_e() függvény leírásánál található.

Menü

A menü kialakítása a kód következő része. Egy felsorolt típusban találhatóak a menügombok sorszámai, hogy később konkrét számok helyett a nevükkel lehessen használni őket a kódban, például a start/stop gomb neve bt_start. Ennek a felsorolt típusnak a módosításával lehet átrendezni a gombok sorrendjét, a funkcióik elrontása nélkül.

Ezalatt található a gombok létrehozása és a menübe (gombok tömbje) helyezése. A menü nulla indexű, illetve utolsó elemének minden szám paramétere -1, így a menü gombjainak például a kirajzolásánál végjelként használhatóak, és nem kell tudnunk a menü gombjainak számát.

A menü működésének további részletei itt találhatóak.

Indítás előkészítése

A menü feltöltése után a dicsőséglista változóinak létrehozása, a lista beolvasása és sorba rendezése történik meg. Végül az SDL használatához szükséges változók is létrejönnek (event, surface, timerID), és megtörténik az SDL felület inicializálása.

while (!quit)

Változók

A játék folyamatos futását a while (!quit) ciklus végzi. A ciklus felett deklarált változók:

- `quit`: Értéke akkor lesz 1, ha a játékból ki kell lépni. Egyébként futás közben az értéke 0.
- `start`: Értéke futás közben 0. 1-es érték esetén elindítja az SDL felület időzítőjét (és így a proton mozgását, utána újra 0 lesz. -1-es érték esetén leállítja az időzítőt, ezzel megállítva a játékot, utána újra 0 lesz.
- `protonreset`: 1-es értéke azt jelenti, hogy alaphelyzetbe kell állítani a protont. Futás közben az értéke 0.
- `mapreset`: A pálya alaphelyzetbe állítására kiadott parancsok 1-es értékre állítják, ennek elvégzése után értéke ismét az alapértelmezett 0 lesz.
- `changelevel`: Pálya váltásakor ebben a változóban van a következő pálya sorszáma.
- `currentlevel`: Az aktuális pálya számát tárolja, például a pontszámok tömbjének indexeléséhez.

A ciklus belseje, a játék lelke

A ciklus első részében található if-ek a játék különböző állapotai közötti váltást szolgálják, indító/megállító, alaphelyzetbe állító funkciókat végeznek a fent leírt változók állapotától függően. Ezután az SDL események feldolgozása történik meg, ennek részei:

Egérgomb lenyomása (SDL_MOUSEBUTTONDOWN)

- Bal egérgomb
  - A menü gombjainak lenyomása (lásd: Button_click() függvény)
  - Az ötödik pálya után megjelenő dicsőséglista gomb megnyomása
- Rácspontra kattintás
  - Bal egérgomb (töltés növelése)
  - Jobb egérgomb (töltés csökkentése)

Egérgomb felengedése (SDL_MOUSEBUTTONUP)

- A kilépés gombra kattintás után
- A dicsőséglista gombra kattintás után
- Menügombra kattintás után
- A fentieken kívül az egérgomb felengedése után minden nem ki/be kapcsolható gomb felengedésre kerül

Billentyű lenyomása (SDL_KEYDOWN)

- A dicsőséglista képernyőn lévő szövegbevitel kezelése
  - A textinput változó 1-es értéke kell a szöveg beírásának detektálásához
  - A kezelt billentyűk a kód kommentjeiben láthatóak
- A játék használata közben használható gyorsgombok kezelése
  - Q, ESCAPE: kilépés
  - R: a proton alaphelyzetbe állítása
  - M: a pálya alaphelyzetbe állítása
  - N: a számok kiírásának ki/be kapcsolása
  - S, SPACE: start/stop gomb
  - T: téma (színséma) változtatása
  - H: dicsőséglista megnyitása
  - 1-5: pálya betöltése
  - NUMPAD+: minden pálya feloldása, tesztelés közbeni használatra

Billentyű felengedése

- Felenged minden nem ki/be kapcsolható gombot

Az időzítő eseményei (SDL_USEREVENT)

- A legelső ellenőrzések megnézik, hogy a proton a pálya szélén van-e, illetve ütközött-e a célponttal.
- Kiszámolásra kerül az eredő erő. A töltések listájának bejárása közben ellenőrzi a program, hogy a proton nem-e ütközött valamelyik töltéssel.
- Ha a proton fallal ütközik, a fal felé mutató sebessége 0 lesz.
- Ha a proton valamelyik sebességkomponensének abszolút értéke nagyobb, mint a vmax változó által megadott érték, a proton sebessége csökkentésre kerül (ez egy kis csalás a tényleges fizikával szemben, de játszható lesz tőle a játék, mivel nem repülhet ki a proton a pályáról hatalmas sebességgel).
- Ezután a proton pozíciója változtatásra kerül a sebessége által meghatározott új helyre.

Kilépés (SDL_QUIT)

- Az SDL felület bezárásakor leállítja a játékot.

Kirajzolás

Az SDL esemény feldolgozása után újrarajzolásra kerül a képernyő, a három frissítő függvény valamelyike által.

A ciklus vége

A játékból való kilépéskor megszakad a while (!quit) ciklus, felszabadításra kerül minden dinamikusan lefoglalt memóriaterület a freememory() függvény által, megtörténik a dicsőséglista visszaírása a megfelelő fájlba, majd végül bezáródik az SDL felület.

Egyszerű struktúrák (pontok.h, pontok.c, vektorok.h, vektorok.c)

Ezek a fájlok a pont és vektor struktúrák definícióit és az ezek alapműveleteit elvégző függvényeket tartalmazzák.

Függvényeik

Vektor Vektor_osszead (Vektor a, Vektor b)

Összead két vektort komponensenként, és visszatér az összegükkel.

double ponttav (double x1, double y1, double x2, double y2)

Két double koordinátákkal átadott pont távolságát számolja ki (Pithagorasz-tétel), és adja vissza.

void Pont_kiir (Pont p)

void Vektor_kiir (Vektor v)

Kiírják a paraméterként kapott struktúra adatait a logba, debugolás céljából.

Megjegyzés a Pont_legyen és Toltes_legyen függvényekhez

Ezekre a függvényekre azért van szükség, mert az alábbi inicializálások nem engedélyezettek:

```
f.a = {fix, h1};
uj->toltes = {x, y, charge, 1};
```

Ezért helyettük ezek a megoldások szerepelnek a kódban:

```
f.a = Pont_legyen(fix, h1);
uj->toltes = Toltes_legyen(x, y, charge, 1);
```

Töltések (toltesek.h, toltesek.c)

Ez a fájl tartalmazza a töltés és mozgótöltés struktúrákat, illetve a töltések láncolt listájának kezeléséhez szükséges függvényeket.

A töltéseket kezelő függvények

Toltes Toltes_legyen (int x, int y, int charge, int editable)

Magyarázatként lásd a fenti megjegyzést.

void Toltes_letrehoz (int x, int y, int charge)

Hozzáfűz egy elemet a láncolt lista végéhez, az alábbi algoritmussal:

- Memóriát foglal egy új ToltesLista struktúrának, majd kitölti annak az adatait. (Ha a foglalás sikertelen, a megfelelő hibakóddal megszakítja a program működését.)
- Ha van már a listában elem, akkor az eddigi utolsó elem következő elemre mutató pointerének megadja az új elem címét, ellenkező esetben az első elemre mutató pointert és az első nem fix elemre mutató pointert irányítja az új elemre.
- Ezután az utolsó elemet jelölő pointert az aktuálisra állítja.
- Végül, ha eddig nem volt nem fix elem a listában, megjelöli az utolsó nem fix töltés utáni elemet (ez a most létrehozott új elem) az első nem fix elemként.

Ez a függvény akkor hívódik meg, amikor a felhasználó kattintása miatt kell létrehozni egy töltést.

void fixToltes_letrehoz (int x, int y, int charge)

Az előzőhöz hasonlóan egy elemet fűz a lista végére, de ez a függvény a pálya beolvasásakor hívódik meg, és nem módosítható töltéseket hoz létre. Az algoritmus nagyon hasonló az előzőhöz, a kivétel csak az, hogy a végén az új töltés lesz az utolsó fix töltésként megjelölve.

void Toltes_torol (ToltesLista *prev, ToltesLista *next)

Kitörli a láncolt lista egy elemét. Ez a függvény akkor hívódik meg, ha a felhasználó művelete miatt kell törölni egy töltést. Paraméterként híváskor a törlendő töltés előtti és utáni elem címét kell átadni, ezen felül pedig a mozgoptr-nek a híváskor a törlendő töltésre kell mutatnia. A különböző szélsőséges esetek kezelését lásd a kód kommentjeiben.

void Toltes_novel (Pont racspont)

void Toltes_csokkent (Pont racspont)

A két függvény majdnem azonos funkciót lát el, de az áttekinthetőség kedvéért nem lettek összevonva. Ezek a függvények hívódnak meg, amikor a játékos rákattint egy rácspontra. A függvények a pont koordinátáit kapják meg.

- Bejárják a töltések láncolt listáját, és ha találnak egy megadott koordinátákkal rendelkező, már létező (és nem fix) töltést, módosítják a töltésének értékét. Ha ezután a töltés abszolút értéke 0 lesz, a töltés törlésre kerül (az előző függvénnyel).
- Ha nem talált meglévő töltést, létrehoz egy új töltést az adott koordinátákon.

void Toltes_rajzol (SDL_Surface *screen, Toltes q, int toltesr, Mozgotoltes proton, int shownumbers)

Kirajzol egy töltést a képernyőre. A töltés az előjelétől függően kap RGB értékeket (itt nem használjuk a szín struktúrát, egyszerűbb csak 3 integerként kezelni őket, lokális változókkal). A nem fix töltések átlátszósága az abszolút értéküktől függ (a gyengébb töltések átlátszóbbak). A nem szerkeszthető töltések fix alpha (a) értékekkel rendelkeznek, és át vannak húzva két vonallal. A menü gombjának állásától függően a nem fix töltések fölé kiírásra kerül a numerikus értékük. A fix töltések értéke mindig meg van jelenítve, mivel anélkül nem lenne a játékos által megállapítható.

A proton, mint mozgó töltés

A proton külön struktúrát kapott, mivel a többi töltéssel ellentétben mozognia kell a pályán.

```
typedef struct {
    double x;
    double y;
    int r;
    int charge;
    Vektor v;
} Mozgotoltes;
```

A többi töltéstől eltérően koordinátái nem csak egészek lehetnek (így egy kicsit szabadabb a mozgása), ezen felül rendelkezik egy sugárral, illetve egy sebességvektorral is.

void Mozgotoltes_rajzol (SDL_Surface *screen, Mozgotoltes p)

A protont kirajzoló függvény.

Falak (falak.h, falak.c)

A falak láncolt listájának kezelése nagyon nagy részben megegyezik a töltések láncolt listájának kezelésével, csak annál helyenként egyszerűbb, így ezeket a függvényeket itt most nem tárgyaljuk.

Falak megadása

A falak kezelésével kapcsolatban érdekes viszont a falak megadása. Egy falat meghatároz két pontja, ezen kívül a Fal struktúrában (falak.h) a fal irányát és egy logikai változót is tárolunk, ami azt fogja megjegyezni, hogy egy adott időpillanatban hozzáér-e az adott falhoz a proton (hogy ezt minden időpillanatban csak egyszer kelljen kiszámolni, erről bővebben itt).

```
typedef struct {
    Pont a, b;
    short dir;
    short touched;
} Fal;
```

A játék csak vízszintes és függőleges falakat kezel, ezért a falakat létrehozó függvények úgy működnek, hogy még véletlenül (sőt, még szándékosan) se tudjunk ferde falakat készíteni velük, emiatt viszont kicsit bonyolultabb a használatuk.
Egy falat az alábbi négy adattal határozunk meg:

- A fal iránya (dir): a vízszintes falat a 0, a függőleges falat az 1 paraméter érték jelöli
- A fal pontjainak fix koordinátája (fix): ez a fal két pontjának közös koordinátája. Vízszintes fal esetén ez az y, függőleges fal esetén az x koordináta.
- A fal végpontjainak nem közös koordinátái (h1, h2): vízszintes fal esetén az x, függőleges fal esetén az y koordináták.

Példák falak megadására

Első példa, a (250, 300) és (250, 500) végpontokkal rendelkező függőleges fal paraméterei:

```
dir = 1, fix = 250, h1 = 300, h2 = 500    vagy
dir = 1, fix = 250, h1 = 500, h2 = 300
```

Azaz a fal két végpontja akár fordítva is megadható.

Még egy példa, az (500, 550) és (850, 550) végpontokkal rendelkező vízszintes fal paraméterei:

```
dir = 0, fix = 550, h1 = 500, h2 = 850    vagy
dir = 0, fix = 550, h1 = 850, h2 = 500
```

A Fal_legyen és Fal_letrehoz függvények ezeket a paramétereket várják bemenetként.

Falhoz érés kezelése

int fuggoleges_falhoz_er (double x, double y, int r)

int vizszintes_falhoz_er (double x, double y, int r)

A játék folyamán szükség van annak az érzékelésére, hogy a proton ütközik-e egy fallal. Ezt a fenti két függvény a falak listájának bejárásával, és az azokkal való ütközés ellenőrzésével végzi el. Az ellenőrzést csak a megfelelő irányú (megfelelő dir paraméterű) falakon végzik el. Ennek eldöntése a következő módon történik (a példa a vízszintes falhoz érés egyik lehetőségét mutatja meg):

```
fal.a.x <= x && x <= fal.b.x
```

Azaz a töltés vízszintesen a fal két végpontja között helyezkedik el (ezt utána fordítva is ellenőrzi a függvény),

```
&& y - r <= fal.a.y && fal.a.y <= y + r
```

és függőlegesen a fal (fix) y koordinátája a töltés legfelső és legalsó pontja között helyezkedik el.

A függvények visszatérési értékei a kód kommentjeiben találhatóak meg.

Ezek a függvények az ellenőrzés után megváltoztatják minden fal touched paraméterét az aktuális állásra, hogy később a rajzolásnál ne kelljen újra meghatározni, hogy egy falhoz hozzáér-e éppen a töltés.

int falban_lenne (int x, int y, int r)

Ezt a függvényt használja a játék annak megvizsgálására, hogy egy játékos által lerakott töltés belelógna-e egy falba. A függvény az előzőekkel megegyező képleteket használ.

Falak kirajzolása

void Fal_rajzol (SDL_Surface *screen, Fal f)

Ez a függvény végzi a falak képernyőre rajzolását. A falak touched tulajdonsága alapján lesznek a kirajzolt falak fehér vagy piros színűek (a fal piros, ha éppen hozzáér a proton). Ez a tulajdonság a fenti két függvény által a main függvényben a proton sebességének kezelésekor újraszámolódik, így rajzolásnál már rendelkezésre áll, és nem kell újra megvizsgálnunk.

Színsémák (szinek.h, szinek.c)

Sémák megadása

A játékban található változtatható témák colorscheme típusú struktúrákban vannak definiálva. Ezek a fájlok ezeket a struktúrákat tartalmazzák, illetve a kezelésükkel foglalkoznak. Egy ilyen struktúra kisebb, color típusú struktúrákból áll, amelyek csupán RGB értékeket tárolnak integerekben.

```
typedef struct {
    int r, g, b;
} color;
```

A colorscheme struktúrák rendelkeznek egy névvel (maximum 9 karakter), ezen kívül pedig a játéktér háttérszínét, a menü alapszíneit, a különböző állapotban lévő gombok háttér- és feliratszíneit, illetve az egyéb szövegek színeit tartalmazzák. Ezek sorrendje és jelölése a struktúrában megtalálható a kódban lévő kommentekben.

Betöltés

A játék alapból tartalmaz négy színsémát, amelyek a következőek: klasszikus szürke, kék, zöld és piros. Ezeken kívül tartalmaz egy random színsémát is, amelyben a színek nincsenek előre meghatározva. A színek betöltését a setaccent() függvény végzi. Ha a paraméterként kapott színséma neve nem "random", akkor a választott színséma előre megadott értékeit tölti be, egyébként pedig az aktuális színséma minden R, G, B értékére "véletlenszerű" színeket állít be.

A játék menüje (menu.h, menu.c)

Gombok

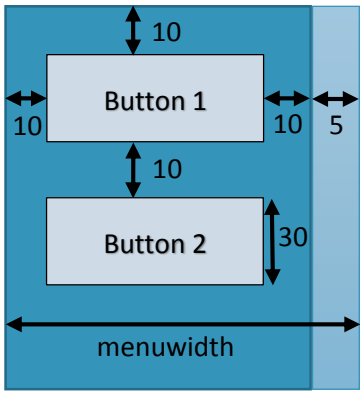
A játék menüje egy gombokból (Button struktúra) álló tömb, ami a main() függvényben található. A struktúra definíciója a menu.h fájlban található:

```
typedef struct {
    int clicked;     /** Eppen lenyomott gomb */
    int enabled;     /** Kattinthato gomb */
    char label[30]; /** A gomb felirata */
    int toggle;      /** Ki/be kapcsolhato gomb-e */
} Button;
```

A gombok színe a fent látható állapotoktól függ. Különböző színűek a nem kattintható és kattintható gombok, az éppen kattintott gombok (vagy ki/be kapcsolható gombok esetén a bekapcsolt gombok), illetve az éppen elérhetővé vált gombok (ez jellemzően a pálya teljesítésekor a következő pálya gombja). Az éppen elérhetővé vált gombokat az enabled mező -1-es értéke jelöli, ami az első kattintáskor 1-re változik.

A menüben lévő gombok pozíciója a sorszámukkal meghatározható, az alább látható módon.

```
int leftedge = 10;
int rightedge = menuwidth - 10 - 5;
int button_top = buttonid * 10 + (buttonid - 1) * 30;
int button_bottom = buttonid * 10 + buttonid * 30;
```



A menü gombjainak pozícionálása

int Button_click (int buttonid, int xpos, int ypos, Button *menu)

Megnézi, hogy a megadott x, y koordináták az adott sorszámú gombon vannak-e (egy adott kattintás a gombon történt-e). Ha a kattintás a gombon történt, és a gomb nincs éppen letiltva (ehhez van szüksége a gombok tömbjére paraméterként), igaz értéket ad vissza.

Kirajzolás

void Button_draw (SDL_Surface *screen, int buttonid, Button button)

Kirajzolja a képernyőre az adott gombot. Ehhez szüksége van a gomb sorszámára a menüben, illetve magára a gombra, hogy az állapota alapján kiválassza a színeit, illetve ráírja a gomb feliratát.

void drawmenu (SDL_Surface *screen, Button *menu)

Megrajzolja a menü hátterét, illetve a menüt a játéktértől elválasztó sávot. Ezek után kirajzolja a menü gombjait.

Névjegy

A játékot készítette Braun Márton Szabolcs, a Programozás alapjai I. elsőéves mérnökinformatikus tárgy nagy házi feladataként, a 2014/2015-ös tanév őszi félévében.

Elérhetőség

Kérdésekkel, észrevételekkel kapcsolatban megkereshet a zsmb13@sch.bme.hu email címen.

Felhasznált források, anyagok

Az SDL multimédia könyvtárat az alábbi linkeken, és az ott linkelt anyagok segítségével használtam fel.
https://infoc.eet.bme.hu/sdl_telepito.php
https://infoc.eet.bme.hu/sdl.php

A feladat a minta nagyházi oldal segítségével készült el.

A játék Code::Blocks 13.12 környezetben készült. A fejlesztés közben előkerülő hibaüzenetek megoldásában a stackoverflow.com oldal segített.

A játék elkészítésének eredeti kiírása

(Grafikus program.) A játék a következő. A program a képernyő adott pontjából, adott irányban kilő egy pozitív töltéssel rendelkező részecskét. A játékos számára adott néhány másik töltés, amelyeket úgy kell elhelyeznie a képernyőn, hogy a mozgó töltés egy megadott célba jusson. Ezek az elhelyezendő töltések lehetnek különböző erősségűek, polaritásúak. Közben a pályán lehetnek fix töltések, falak stb. (Hasonló játék itt: http://kmk.blog.hu/2007/07/26/newton_kedvenc_jateka)

A program számoljon pontszámot a játékos számára (pl. hány próbálkozásra sikerült megoldania a pályákat), és ez alapján tartson nyilván dicsőséglistát is, amelyet fájlba ment és vissza is olvas! A programnak tetszőlegesen sok töltést kell tudni kezelnie. A pályák leírását (hol van töltés, hol van fal) olvasd fájlból! Egy pályán lehessen tetszőlegesen sok mind a kettőből!