Töltés játék – programozói dokumentáció

1. Adatszerkezet

1. Player (playerHandler.c)

A játékost egy struktúrában tárolom, hiszen több olyan adatot kell tárolni, ami szorosan kapcsolódik hozzá.

- A *name* egy 50 hosszú string, ennél hosszabb nevet nem engedélyez a program.
- A *score* tárolja a játékosnak az aktuális pontjait
- A *level* azt tárolja, hogy a user éppen melyik pályán van
- A *polarity* azt tárolja, hogy a játékos éppen vonzó, vagy taszító részecskét szeretne lerakni
- A *started* jelzi, hogy a játékos elindította-e a töltött részecskét

```
typedef struct Player
{
    char name[50];
    int score;
    int level;
    int polaritiy;
    bool started;
} Player;
```

1.1 PlayerList(playerHandler.c)

A játékosokat egy egyszeresen láncolt listában tárolom, itt csak két mező van a Player, illetve természetesen a pointer.

2. Charge (mapHandler.c)

A töltéseket is egy struktúrában tárolom.

- Az *x,y* koordináta, melyek egész típusúak.
- *Weight*, ami a töltés súlyát tárolja (lényegében mindegy, hogy bolygóként kezeljük vagy töltésként, a fizikája ugyan az).

```
typedef struct Charge
{
    int x;
    int y;
    int weight;

} Charge;
```

3. Wall (mapHandler.c)

A falaknak is van egy külön adat szerkezetük.

- Itt is megvan az x,y koordináta, ami a bal felső sarkát jelöli a falnak
- A w a width-re utal, ami a szélessége lesz a falnak
- A h a height, a magassága a falnak.

```
typedef struct Wall
{
    int x;
    int y;
    int w;
    int h;
} Wall;
```

4. **Map (mapHandler.c)**

A legnagyobb adatszerkezet eddig. Ez a struktúra már az előzőkre épít.

- *startPointX*: A pályán a fő töltés kezdőpontjának X koordinátája
- *startPointY*: A pályán a fő töltés kezdőpontjának Y koordinátája
- *endPointX*: A pályán a célnak az X koordinátája (mindig 35px széles)
- *endPointY*: A pályán a célnak az Y koordinátája (mindig 35px magas)
- *startVelocityX*: A kezdő sebességvektor X koordinátája
- *startVelocityY*: A kezdő sebességvektor Y koordinátája
- *chargeCount*: A lerakott töltéseket hivatott számlálni
- *wallCount*: A pályán lévő falak számát határozza meg
- *walls*: Egy Wall struktúrára mutató pointer, ez egy dinamikus tömb, ebben vannak tárolva a falak
- *charges*: Egy Charge struktúrára mutató pointer, szintén dinamikus tömb, amiben a töltések vannak tárolva

```
typedef struct Map
{

    int startPointX;
    int startPointY;
    int endPointX;
    int endPointY;
    int startVelocityX;
    int startVelocityY;

    int chargeCount;
    int wallCount;
    Wall* walls;
    Charge* charges;

} Map;
```

5. **MainCharge (mapHandler.c)**

- *x,y*: A fő töltés X és Y koordinátája
- *xVelocity*: A fő töltés sebességvektorának X koordinátája
- *yVelocity*: A fő töltés sebességvektorának Y koordinátája
- *weight*: A fő töltés súlya.

```
typedef struct MainCharge
{
    double x;
    double y;
    double xVelocity;
    double yVelocity;
    double weight;
} MainCharge;
```

A későbbiekben a játékosokat egy láncolt listában fogom tárolni.

2. Fontosabb függvények

1. graphics.c

void drawWall(SDL_Surface* screen, Wall wall)

A függvény kirajzol egy falat a pályára.

void drawEndPoint(SDL_Surface* screen, Map map)

Kirajzolja a végpontját a pályának.

bool checkCollision(MainCharge* mainCharge, Wall wall)

Ellenőrzi, hogy a fő töltés ütközik-e egy fallal. Ha igen, akkor igazzal tér vissza. A hívó felelőssége, hogy megfelelő sűrűséggel hívja meg és az összes falra.

void initializeMap(SDL_Surface* screen, Player player, Map map)

Felrajzolja a pályát az SDL surfacere. A játékos started állapotát megváltoztatja falsera, mivel a játékos a pálya inicalizálásakor nem indult még el. Illetve a mapban megadott értékekre kirajzolja a fő töltést, illetve a végpontot, és ha vannak a falakat.

void tick(SDL_Surface* screen, Player* player, Map* map, MainCharge* mainCharge)

A függvény feladata, hogy az SDL időzítő által pusholt user eventekre mindig lefusson és elvégezze a szükséges számításokat, lépéseket. Itt van ellenőrizve, hogy a játékos meghalt-e, ütközik-e fallal, nyert-e. A töltések itt hatnak a fő töltésre.

bool chargePlaceValidation(Map map, int x, int y)

A függvény egy egyszerű segédfüggvény, ami azt nézi meg, hogy a játékos a töltést nem a végpontra akarja-e rakni, vagy egy falba. Ha nem ezekre akarja rakni, akkor trueval tér vissza a függvény.

void initGraphicsWindow(Player player, Map map)

Itt nyílik meg a grafikus ablak, illetve itt vannak kezelve az SDL eventek.

2. **mapHandler.c**

bool addCharge(Map* map, Charge charge)

Egy töltést ad hozzá a maphoz, és trueval tér vissza, ha sikerül.

Void freeCharges(Map* map)

Felszabadítja a maphoz tartozó töltésnek lefoglalt memóriaterületet

void deleteCharge(Map* map)

Egyszerűen levon egyet a map charge számlálójából. Szándékosan nem foglalom újra a területet, mert ez fölösleges lenne, inkább, ha ez a számláló eléri a nullát, akkor felszabadítom az egész lefoglalt területet. Ez a hívó felelőssége, hogy ha 0-ra hívná ezt a függvényt, akkor szabadítsa fel a map->charges változóban megcímzett lefoglalt területet.

Void freeWalls(Map* map)

Felszabadítja a maphoz tartozó falak számára lefoglalt területet.

Map loadMap(char mapName[])

Betölt fileból egy mapot és vissza is tér vele. Egy stringet vár: „mapX.txt” néven, ahol X a pálya számát jelöli.

3. **playerHandler.c**

bool savePlayer(Player player)

Elment egy játékost file-ba a „highscore.txt”-be. Egyelőre csak egy új sorba hozzáfűzi.

Bool sortPlayers(PLayerList* firstPlayer)

A feladata, hogy a highscore.txt-ben tárolt usereket újra írja a paraméterként megadott láncolt lista sorrendjében. Itt fontos szerepet játszik majd a loadPlayers() függvény, amelyik sorrendben készíti el a láncolt listát, így az azáltal elkészített lista alkalmas erre a függvényre.

PlayerList* loadPlayers()

A függvény a highscore.txt-ben tárolt usereket láncolt listába betölti a memóriába. A sorrendet pontok szerint csökkenő sorrendben tölti be. Tehát a lista első tagja a legnagyobb pontú user.

Void freePlayers(PlayerList* firstPlayer)

A loadPlayers() függvény párja, ami a lefoglalt láncolt listát bejárja és felszabadítja az összes node-ot.

Player initPlayer()

Létrehoz egy új játékost. Az alap értékeit beállítja, illetve a nevét bekéri consoleból.

void playerWins(Player* player)

Azt az eseményt kezeli, amikor a játékos nyer. Új pályára teszi, az indított állapotát falsera állítja, illetve elmenti a játékost.

3. Parancsok a játékban

'R': Reset. Alaphelyzetbe állítja a pályát és levon 10 pontot.
'O': Teszt funkció, a következő pályára léptet.
'1': Negatív töltésű részecske kiválasztása
'2': Pozitív töltésű részecske kiválasztása
'SPACE': A játék indítása
Bal egérgomb: Töltés lerakása
Jobb egérgomb: A legutoljára lerakott töltést kitörli.
'Q': Utolsó pályán a játékból kilép, ha sikeresen megnyerte a felhasználó.

4. Szükséges könyvtárak

Az SDL 1.0 könyvtárak szükségesek a program futásához, ezeknek a DLL fáljait a projekt mellé be kell másolni, vagy egy globális változót rámutatni a DLL fájlokat tartalmazó mappára.

5. A projekt felépítése

a. **Main.c**
   A program konzolos része itt van megvalósítva, egy menü van implementálva.
b. **Graphics.c**
   A grafikai függvények vannak ebben a modulban. Bármilyen számítás, ami a grafikai elmekhez kapcsolódik szintén itt található.
c. **mapHandler.c**
   A pálya tulajdonságait befolyásoló függvények találhatók itt, pl.: a falak elhelyezkedését, töltések számát lehet változtatni itt.
d. **playerHandler.c**
   A játékos tulajdonságait befolyásoló függvények vannak itt, pl.: a játékos pontjai stb.

6. TXT fájlok formátumai

A játék pályáit és játékosait TXT fájlban tárolom. Ezeknek a formátuma a következő:

1. mapX.txt
   Az X egy tetszőleges szám, ez határozza meg azt, hogy hányadik pálya. Az első sor tartalmazza a fő töltés kiindulási X Y koordinátáit, majd a cél négyzetnek a bal felső sarkának X Y koordinátáit. A következő sor a fő töltés kezdő sebesség vektorát tartalmazza. Minden további sor egy falat reprezentál a következő képpen:
   X Y W H, ahol X,Y: a fal bal felső sarkának koordinátái, és W,H: a szélesség, magasság.
2. Highscore.txt
   Ez tartalmazza a felhasználókat, mégpedig a következő képpen:
   Username (sortörés)
   UserScore (sortörés)