Programozói dokumentáció - My-Awesome-Game

Pintér Tamás
JY4D5L

.c/.h:

main, global, textures, player, ball, bullet, button, menu, scores, settings, game, debugmalloc

Textures mappa //ebben van az összes felhasznált textúra

scores.txt //itt vannak elmentve a toplista eredmények

Felhasznált könyvtár: raylib.h

https://github.com/raysan5/raylib - cheatsheet

GLOBAL.H

```
int resolutionX, resolutionY //az alapértelmezettnek vett képernyőfelbontás (1920x1080)

int screenWidth, screenHeight //játék közben az ablak felbontása

int menu_screenWidth, menu_screenHeight //menüben az ablak felbontása
```

BUTTONS.C

```
void setupButtons(); //az összes gomb és kattintható elem pozíciója és mérete itt van definiálva
```

SETTINGS.C

```
void settingsButtonClick(); //ez a függvény figyeli, hogy rákattintottunk-e valamelyik gombra, és ha igen, akkor beállítja a megfelelő nehézségi szintet, és hátteret

void renderSettings(); //rendereli a beállítások menü tartalmát. Ha valamelyik gomb ki van választva, akkor egy keretet rak köré

void initGameData(); //beállítja a nehézségi szint alapján a labdák, lövedékek, és élet pontok számát
```

TEXTURES.C

```
void loadImage(); //betölti a memóriába a képeket, formatálja, majd Texture2D típusú változóba rakja őket
```

TEXTURES.H

```
Image textures[49]; //lefoglal egy 49 méretű tömböt a textúráknak

Texture2D …; //az itt felsorolt textúrákat lehet használni a továbbiakban
```

PLAYER.C

```
// a játékost egy struktúrában tárolom el, így könnyen és egyértelműen lehet hivatkozni különböző tulajdonságaira

void setupPlayer(); //a játékos pozíciója, sebessége és mérete itt van definiálva. A játékos egy Player típusú player struktúra, ami a player.h-ban lett létrehozva

void movePlayer(); //a bal illetve jobb nyilakkal mozgathatjuk a játékost

void renderPlayer(); //a háttértől függően rendereéi be a játékos figurát
```

PLAYER.H

```
typedef struct Player{double xpos; double ypos;
int xsize; int ysize; int speed;}Player; //itt vannak definiálva a játékos tulajdonságai

bool player_isAlive; //ezzel tudom változtatni, hogy a játékos még él-e, vagy sem
```

MENU.C

```
void renderMenu(); //rendereli a főmenü hátterét, ami egy mozgó napból áll, egy háttérképből, meg a gombokból

bool isOverButton(Button button); //figyeli, hogy az adott gomb felett van-e az egerünk

void menuButtonClick(); //azt nézi, hogy melyik gombra kattintottunk. Ha a startra, akkor elindítja a számlálót, és a játékos adatait beállítja. Ha a settingsre, akkor átváltja a játék állapotát a SETTINGS-re. Ha a scoresra, akkor pedig a SCORES állapotra.

void wait_sec(int timeDiff, int time); //a megadott számtól visszaszámol, és a végén átváltja az állapotot a GAME-re

void renderButtons(); //rendereli a menü gombjait, és ha nem volt még nehézség vagy háttér választva, akkor hibaüzenetet ír ki a képernyőre.
```

MENU.H

```
typedef enum Difficulty {DIFFICULTY_UNSET, EASY, MEDIUM, HARD}Difficulty; //a játék nehézségét ebben a struktúrában tárolom

typedef enum state{MENU, GAME, SETTINGS, SCORES, END} state; //a játék jelenlegi állapotának követésére szolgál

typedef enum Backgrounds{BACKGROUND_UNSET, FOREST, MOUNTAINS, SPACE, JAPAN}Backgrounds; //az aktuális hátteret határozom meg vele
```

SCORES.C

```
struct topScores[10]; // ebben a struktúrában vannak elmentve a ranglista elemei

void renderTime(Vector2 position, int time, int fontSize); //átkonvertálja az eltelt időt óra:perc formátumba, és kiírja a képernyőre 00:00 formában

void renderScores(Score scoreArray[10], int x, int y, int fs, int textSpace); // az x, és y koordináták segítségével kirajzolja a ranglista adatait, amit a scoreArrayből kapunk meg

void writeToFile(Score scoreArray[10]); //elmenti a ranglistát a scores nevű txt filreba

void readFromFile(Score scoreArray[10]); //visszatölti a ranglistát a scores nevű txt fileból

void arraySort(Score scoreArray[10]); //rendezi a ranglistát pontszám szerint csökkenő sorrendbe

void renderDifficulty(Vector2 where, int fontSize); //kiírja a nehézséget a ranglistára

void updateScores(Score scoreArray[10], int number);
     //eldönti a számról, hogy a ranglistára való-e, és ha igen, akkor belerakja

void resetLeaderboard(Score scoreArray[10]); //lenullázza a pontokat a ranglistán

void endScreenButtons(); //rendereli a játék vége képernyőn a gombokat

void renderEnd(); //rendereli a játék vége képernyőt

void endOfGame(Score scoreArray[10]); //a játék vége ciklus

void renderScoresMenu(); //rendereli a scores képernyőt
```

```
void scoresMenuButtons(Score scoreArray[10]); //rendereli
a scores menü gombjait

void scores(Score scoreArray[10]); //scores ciklus
```

MAIN.C

```
//az elején elindít néhány függvényt, amik a játék
futásához és alapbeállításaihoz kellenek, ez után pedig
egy állapotgép segítségével lehet változtatni a játék
állapotát. Ez a már bemutatott gameState enum adataival
dolgozik, amiket a különböző játékciklusok végén
változtat meg. Az állapotváltás után mindig beállítja az
ablak méretét, és a kurzort elrejti szükség esetén. A
játék végén felszabadítja a szükséges dolgokat (két
láncolt lista, és egy vagy két tömb) Amikor ezekkel is
megvan, akkor bezárja az ablakot, és vége a játéknak.
```

BALL.H

```
typedef struct Ball{

     double xpos,
     double ypos,
     double vy,
     double vx,
     double gravity,
     double bounce,
     int radius,
     int HP,
     bool visible,
     struct Ball *next

}Ball; //a labdák tulajdonságait ebben a láncolt listában
tárolom
```

BALL.C

```
// A pattogó labdákat láncolt listában tárolom, mert
szerintem így a legegyszerűbb a kezelésük. Könnyű
belerakni és kivenni belőle tagokat, és mivel a játék
arról szól hogy le kell lőni a labdákat, emiatt ez egy
elég erős választásnak tűnt. Ez a file javarészt láncolt
listákat kezelő, és változtató függvényekből áll. A játék
kezdetekor véletlenszerűen vagy bal, vagy jobb oldalra
lerak egy megadott mennyiségű labdát bizonyos
tulajdonságokkal, majd amikor a játékos kilő egyet, akkor
egy újat rak a helyére.

Ball *balls = NULL; //létrehozok egy Ball típusú láncolt
listát, ami segítségével fogom majd a labdákat kezelni
```

```
int ballNumber; //különböző nehézségekkel különböző mennyiségű labda van a pályán, ez azt tárolja, hogy éppen mennyinek kell pályán lennie

int ballNumber_current = 0; //a jelenleg pályán lévő labdák száma

void freeList_ball(); //felszabadítja az összes labdát

Ball *freeBalls_dead(Ball *head); //felszabadítja a már kilőtt labdákat

Ball *list_append_ball (Ball *head); //a lista végére szúr egy labdát, beállít neki egy véletlenszerű nagyságot, és le is spawnolja a képernyő bal vagy jobb oldalára véletlenszerűen (a spawn függvény akkor hívja meg ezt a függvényt, ha éppen spannolni kell labdát, így lesz egyből lerakva a pályára)

void spawnBall(); //ha egy labda ki lett lőve, akkor a függvény meghívja a lista végére fűző függvényt, és megnöveli a jelenleg a pályán lévő labdák számát

void ballBounce(Ball *head, bool gravity); // a labda pattogásáért felel ez a függvény, és hogy ha a labda túlrepülne a falon, akkor visszakerüljön azonnal a pályára.

void renderBalls(); //végigmegy a labdák listáján, és kirajzolja a megfelelő adatokkal a labdákat, majd rájuk írja az életüket

void collisionWall(Ball *head); //ha a labda a falnak ütközik, akkor vissza is pattan

void applyPhysics_Balls(Ball *head); //ha a játékos még életben van, akkor pattogtatja a labdákat, és figyeli, hogy visszapattanjanak a falról

void updateBalls(Balls *head); //ha a labdák élete lecsökken bizonyos értékek alá, akkor a méretüket és textúrájukat változtatja annak megfelelően
```

BULLET.H

```
typedef struct Bullet{

     double xpos, // a lövedék x pozíciója
     double ypos, // a lövedék y pozíciója
     bool visible, // a lövedék látható-e éppen vagy sem
     struct Bullet *next // a lista következő eleme

}Bullet; //a lövedékek tulajdonságait ebben a láncolt listában tárolom
```

BULLET.C

```
// Ugyan úgy mint a labdáknál, itt is a láncolt listákra esett a választás. Szintén azért praktikus, mert folyamatosan kell felszabadítani a lövedékeket, és újakat rakni utánuk. Új lövedékeket soronként rakok le, így ilyenkor egy sornyi lövedékkel bővül a láncolt lista

Bullet *bullets = NULL;

typedef struct BulletProperties{

     clock_t shoot; //a lövedékek késleltetésére van
     int shoot_delay; //késleltetés mértéke
     int bulletCount; //egy sorban lévő lövedékek száma
     int bulletRadius; //lövedékek mérete
     int bulletSpeed; //lövedékek sebessége
     int bulletDamage; //lövedékek sebzése

}BulletProperties; // ezek az összes lövedékre vonatkozó tulajdonságok

void freeList_bullet(); //felszabadítja az összes lövedéket

Bullet *freeBulletsOutside(Bullet *head); //felszabadítja a pályát elhagyó lövedékeket

Bullet *list_append_bullet(Bullet *head, double x, double y); //hozzáfűz a listához egy lövedéket a megadott x és y koordinátákkal

void spawnBullets(); //a játékos pozíciójától függően lerak annyi darab lövedéket egy sorba egymástól egyenlő távolságra, amennyi be van állítva a bulletCount változóban

void updateBullets(); //a bulletSpeed változó értékével változtatja a lövedékek pozícióját, így azok előre haladnak folyamatosan
```

```
void renderBullets(); //berendereli a lövedékeket a háttértől függő textúrával
```

GAME.H

```
typedef enum StopGame{PAUSE, RESUME}StopGame;
```

GAME.C

```
Vector2 position = {10.0f, 30.0f};
Rectangle frameRec = {0.0f, 0.0f, 140/4, 30};
int curretFrame;
int framesCounter;
int frameSpeed; //ezek a bal felső sarokban lévő szív animációjához kellenek

double **endGame = NULL; //ez az a tömb, amiben tárolom a játék megállításához és ugyan onnan folytatásához szükséges adatokat

void setupBackupArray(); //lefoglalja a megfelelő nagyságú területet a fenti tömbnek

void freeBackupArray(); //felszabadítja a fenti tömböt

void stopGame(StopGame stopTheGame); //ha PAUSE paraméterrel hívjuk meg, akkor lementi a pályán lévő labdák, lövedékek, és játékos adatait, és mikor meghívjuk RESUME paraméterrel, akkor azokkal folytatja a játékot

bool playerBallCollision(Ball *head); //visszaadja, hogy a játékos és valamelyik labda ütközött-e

void playerLife(); //a játékos labdával való ütközésekor levon egyet a játékos életéből, ha pedig nem maradt már mit levonni, akkor kiírja a GAME OVER feliratot a képernyőre

void pause_resume(); //X gomb lenyomása esetén megállítja, Y esetén pedig folytatja a játékot

void BulletBallCollision(Ball *ball_head, Bullet *bullet_head); //ha a lövedékek eltalálják valamelyik labdát, akkor annak az életéből levonja a sebzést, továbbá figyeli, hogy él-e még az adott labda vagy sem, és ha nem, akkor a láthatóságát hamisra állítja, ami következtében már nem lesz se renderelve, sem figyelve, csak hamarosan felszabadítva. E mellett változtatja az éppen pályán lévő labdák értékét.
```

```
void onDamageAnimation_heart(); //ha a játékost eltalálja egy labda, akkor lejátszik egy animációt

void renderBackground(); //rendereli a hátteret, és a többi dolgot a képernyőn (életerők számát, a szívet, és a feliratot)

void game(); //a játék ciklus az elején setupolja a játékost, beállítja a szükséges adatokat, inicializálja a játékot, aztán belép a while loopba. Minden lefutás végén ellenőrzi, hogy volt-e kilépési feltétel, és ha volt, akkor felszabadítja a labdákat, a lövedékeket, és leállítja a játékidő számlálását.
```