Puyo Puyo programozói specifikáció

Indítás előtt

Fordítás

A forrásfájlokat a csatolt Makefile-ban szereplő paranccsal fordítom.
A fordítási parancs: `gcc -I src/include -L src/lib -o main *.c -lmingw32 -lSDL2main -lSDL2 -lSDL2_gfx -lSDL2_ttf -lSDL2_image -lSDL2_mixer -mwindows -Werror -Wall -pedantic`
Az innen keletkező `main.exe`-t kell futtatni.
A keletkező `main.exe`-vel egy szinten kell lennie a csatolt `CONSOLAB.TTF` fájlnak, ami a betűtípust tartalmazza.
A hibákat a `log.txt` fájlba írja. A debugmalloc kimenete nem jelenik meg, a `main.c`-ben egy sor visszakommentelésével a `debugmalloc_log.txt`-be írja a kimenetet.

Működés

Megjegyzés: Általában a `-1`-es vagy `false` visszatérési érték jelenti, ha hiba történt. A pontos jelentés a függvények előtt kommentben szerepel.

`main.c`

Inicializálja az SDL-t, valamint a program végén leállítja.

`sdl_setup.c`

Itt vannak az SDL indításához és leállításához szükséges függvények, valamint az `input_text()` is, amivel a szöveget lehet beolvasni. Az itt lévő kód kis változtatásokkal az InfoC oldalról származik.

`menu_selector.c`

Ez a fájl kezeli a különböző képernyők közti váltást, a header file-ban vannak a játékhoz használt adattípusok.
A `menu_selector_loop()` függvényben lévő ciklus meghívja az egyes képernyők `loop()` függvényét, amik az SDL eseményeket kezelik. A `loop()` függvények visszatérési alapján választja ki, mi legyen a következő képernyő.

- `-1`: Hiba történt, program leáll

- `0`: Játékos kilépett X-el
- `1`: Játékképernyő
- `2`: Beállítások
- `3`: Ranglista
- `4`: Játék vége képernyő

Adatstruktúrák

- `Block`: Egy cella lehetséges értékei
- `Piece`: Két blokk, azaz egy rész, koordinátákkal. A mozgatott (aktív) rész ezt használja.
- `ScoreData:` Pontszám információi.
- `GameState`: Egy játékállás összes információja.
  - `GameState.board`: A játéktábla, a lerakott blokkokat tartalmazza.

`game_loop.c`

- `game_loop()`: az SDL események kezelése, többi függvény meghívása.
- `gravity_timer()`: A rész magától lefelé mozgásának eseményeit hozza létre. A `real_ms`-ből kivont értékkel lehet állítani, milyen gyorsan gyorsuljon a gravitáció.
- `egyéb timer()`: Hosszú lenyomásnál az akció ismétléséért felelősek.
  Az időzítők saját USEREVENTEKET használnak, a forráskód alján szerepel, melyik mit jelent.
- `move_left(), move_right(), soft_drop(), hard_drop(), natural_gravity()`: A névnek megfelelő lépés elvégzése az aktív részen
- `lock_active_piece()`: Lehelyezi az éppen aktív részt a táblára. Elvégzi a gravitációs és törlési lépéseket, frissíti a pontszámot.
- `pop_groups()`: Kitörli a négy, vagy nagyobb csoportokat, valamint le is játszik egy törlési animációt.

`piece_rotations.c`

Itt szerepel a két fajta forgatás függvénye. Mindkettő bonyolult, mivel minden esetet külön kell kezelni. Cserébe a rész nem tud lelépni a tábláról, és ha olyan helyre forogna, ahova nem tud, megpróbálja magát elrúgni, hogy sikerüljön a forgatás. Felfelé rúgni a végtelen játékok elkerülése miatt csak 3-szor lehet.

`settings_loop.c`

Kezeli a beállítások menüt.
A `settings_loop()` kezeli az SDL eseményeket. Itt változik meg a tábla szélessége és magassága.

`leaderboard_loop.c`

Kezeli a ranglista menüt.
Megjeleníti a kiválasztott sorrend alapján az elmentett eredményeket.
A `leaderboard_loop()` kezeli az SDL eseményeket.

A `sorted_ids[3][_]`-ben szerepel a megjelenítés sorrendje. Az tömb elemei a különböző statisztika alapján szortírozott ID tömbök.
PL a `sorted_ids[0]` a pontszám alapján szortírozott tömb. Az éppen megjelenített szortírozást a `sorted_id` tárolja.
Ha kevesebb, mint 15 elem van, akkor a maradék értékek `-1`-el lesznek feltöltve.

Függvények

- `sort_entries()`: Előállítja a `sorted_ids[3][_]`-t, azaz mind a három fajta szortírozási sorrendet.
- `render_entries()`: Kirajzolja az eredményeket a megfelelő sorrendben.

`game_over_loop()`:

Megjeleníti a játék vége képernyőt, beolvassa a játékos nevét `input_text()`-el, majd ha kell, elmenti azt.

`render.c`

Itt szerepelnek a több helyen is használt rajzoló függvények. A header file-ban definiálva van pár szín, mint `SDL_Color`-ok. Rajzolásnál nem használom fel, hogy `SDL_Color` típusúak, hanem kiveszem az egyes r, g, b, a, mezőket a felhasználáskor.

Adatstruktúra

- `CommonRenderData`: Összefoglaló típus, ami tartalmazza a szükséges adatokat egy rajzoláshoz. Minden rajzoló függvény ezt kéri be, akkor is, ha csak a `renderer` és `font` mezőket használná. Ezeken kívül a játéktábla méreteit és pozícióját tárolja.

Függvények

- `render_game()`: Kirajzolja a játékállást, a soron következő részeket, és a pontszámot.
- `render_board()`: Kirajzolja a táblát és a lerakott blokkokat.
- `render_block_on_board()`: Kirajzol egy blokkot a tábla adott mezejére, a megadott méretszorzóval. Csak a törlési animáció és az aktív rész használja, mert a tábla kirajzolásához összekötő vonalakat is kell rajzolni.

- `render_text_block()` : Kiír egy szöveget választható keretszínnel.

`database.c`

A `scores.txt` állományt kezeli.

Adatstruktúrák

- `Entry` : Egy elmentett adatsor. Játékos neve, pontszámai, és a tábla mérete.
- `Entries` : Dinamikus `Entry` tömb, a méretével egybecsatolva. Ebbe olvasható be az összes elmentett adat. Használat végén fel kell szabadítani a `array` mezejét.

Függvények

- `new_entry()` : Lefoglal és visszaad egy üres `Entries` tömböt.
- `read_entries()` Beolvassa a `scores.txt` -ből az elmentett eredményeket egy `Entries` -be.
- `insert_entry()` Beilleszt egy eredményt a `scores.txt` -be. Ha már szerepelt a név, felülírja az eredményeket.