Parancssoros Tetris játék

- Programozói dokumentáció

> Készítette: Schelb Arnold - S761AX
>
> Laborvezető: Sinkovics Bálint

Projekt felépítése

Modulok

A program 4 modulból épül fel, ezek a file-ok a:

1. `main.py`: Futtatásával indítható a program, innen hívódik meg a program többi része is.
2. `menu.py`: Tartalmazza az elsődleges felhasználói felületet, a különféle menüket, és az azokhoz kapcsolódó függvényeket/objektumokat.
3. `control.py`: Benne találhatóak a program játék közbeni irányításához és kezeléséhez szükséges függvények.
4. `draw.py`: A kijelzőre való kirajzolásért felel, elsősorban játék közben, illetve a menüknél a logók rajzolásáért.

Generált file-ok

- `save.txt`: Mentésnél létrehozott szöveges file, üres sorokkal elválasztva tartalmazza a beolvasott játék indításához szükséges adatokat.
- `highscores.txt`: Dicsőséglista, amely soronként az elmentett legjobb 5 pontszámot tárolja, felhasználó által adott nevekkel.

Környezet

A program futtatásához szükséges a pyconio (és közvetve a colorama) modul, amiket külön telepíteni viszont nem szükséges, a `pyconio/`, (illetve a `colorama/`) mappák már tartalmazzák azokat. Az ezeken kívül használt modulok szabványos, a Pythonba alapértelmezetten beépített modulok.

Adatszerkezetek

1. *field* - Pálya (mátrix):
    - 2 dimenziós lista, ami a pálya jelenlegi állapotát tárolja.
    - Mérete megadható a felhasználó által. Alapértelmezetten a sorok száma 20, a sorok:oszlopok aránya pedig mindig 2:1.
    - A listán belül listák elemei alapesetben 0-ák, ha egy tetromínó pedig az adott cellába ér és rögzül, az általa elfoglalt hely mentésre kerül. Ilyenkor az elfoglalt cellák tartalma felülíródik a tetromínót szimbolizáló betűvel.

        > A pálya tárolásához ez az adatszerkezet tűnt a legmegfelelőbbnek, mert az indexek koordinátáknak feleltethetők meg, illetve ábrázolva is hasonlít a 2D-s lista a játéktérhez.

2. *Shape* - Betűk tetromínóhoz (alakhoz) rendelése:
    - Egy segédosztály, amely a megadott betűhöz (I, J, L, O, S, T, Z közül) egy 2 dimenziós listát rendel (*units*) ami a tetromínó kirajzolásánál szükséges egységeket tartalmazza (0 ha ki kell hagyni, 1 ha meg kell jeleníteni, új sornál pedig új lista).

        > Itt is a 2D lista tűnt a leghelyesebbnek, a fenti okok miatt is, és mert könnyen reprezentálható, ha egy egység "kimarad" az alakban, vagy új sor következik.

3. *Tetromino* - Játékban használt elem, a szükséges adatokkal:
    - self.*units*: A létrehozásnál megadott betűhöz a *Shape* által rendelt 2D listát adja.
    - self.*pos*: A tetromínó (bal felső unit-jának) pozíciója egy listában, sorban X, és Y koordinátákkal.

        > A mutábilitás miatt választottam listát a pozíciónak.

    - self.*color*: A pyconio modul színei közül egy, ami megfelel a klasszikus Tetris színeinek, és a get_color() függvénnyel kérdezhető le, paraméterként az egyik fentebb felsorolt betűt adva.

        > Ezt használja a pálya rajzolásának függvénye is, ezért volt célszerű külön függvénybe szervezni, és nem a konstruktorba építeni.

    - self.*shape*: A tetromínó alakját reprezentáló betűt tárolja, ami a létrehozásnál is meg lett adva.

> Erre a tetromínó "módosításánál" van szükség, amikor létre kell hozni egy újat, hasonló tulajdonságokkal, és először azon elvégezni a kellő módosításokat. Például mozgatásnál, hogy leellenőrizzük, a mozgatás után is még a pályán belül lenne-e az adott tetromínó.

4. *Button* - Menüben használt gomb:
    - self.*text*: A gomb szövege, ami a képernyőre is kiírásra kerül, és konstrukciónál is szükséges paraméter.
    - self.*function*: A gomb funkciója, azaz, hogy kiválasztásnál mit hajt végre, létrehozásnál szintén szükséges.

        > Ez is egy sztring, a könnyebb átláthatóság, és sokféle érték miatt.

    - self.*active*: Alapértelmezetten False érték, azt jelzi, hogy az adott gomb épp ki van-e választva.
5. *Options* - Beállítások tárolására használt objektum:
    - self.*size*: Alapértelmezetten 20, a felhasználó által megadott pályaméretet tárolja.
    - self.*level*: Alapértelmezetten 1, ez tárolja a kezdeti nehézségi szintet.

        > A beállítások tárolásához azért választottam objektumot, mert azt a függvények módosítani tudják, nem úgy mint a lokális változókat.

6. *Score* - Toplista bejegyzés adatainak tárolásához:
    - self.*name*: A felhasználó által megadott név.

        > Nem tartalmazhat ":" karaktert, mert a toplistát file-ba mentve azt használjuk a név és a pontszám határolására is.
        > A program ugyanakkor lekezeli, ha a felhasználó mégis ír ":"-ot a nevébe.

    - self.*points*: A játékos által elért pontszámot tárolja, integerben.

Függvények

main.py

1. main()

- Meghívja a menu modul *main_menu()* függvényét, és ha a játékos a játék indítását/betöltését választja, meghívja a *game_init()* függvényt, egyébként pedig (kivéve Windowson) kirajzolja az elrejtett kurzort, és visszatér.

2. game_init(mode, fieldsize, level)
    - Az adott *mode* alapján inicializálja a játékot, ha a *mode* `new`, új pályát és elemet generál, ha pedig `load`, betölti azokat a `save.txt` file-ból. A függvény lekezeli a nem található file esetét is.
3. mainloop(tetro, field, next, points=0, level=1)
    - Kirajzolja a pályát és a jelenleg aktív tetromínót, a megfelelő helyre.
    - Inicializálja az *ingame* tuple-t, True és 0 értékekkel, amik sorban azt jelzik, hogy a játék folyamatban van-e, illetve hogy mennyi a pillanatnyi pluszpontszám, amit a ciklus végén hozzá kell adni az összpontszámhoz.
    - Fő része a `while ingame[0]` ciklus, ami alatt a játék vezérelhető.
    - Része emellett a többi, játék-specifikus ellenőrzés (ütközések, időzítés, játék vége, szintlépés).

menu.py

1. main_menu(settings=None)
    - A függvényben listába felsorolt gombokkal visszatérési értékként meghívja a *menu()* függvényt, ha pedig beállítások már meg lettek adva, akkor azokkal együtt.
2. pause()
    - A megállítás menü gombjaival szintén a *menu()* függvényt hívja visszatérési értékként.
3. save_menu()
    - Mentés utáni menü, a fentiekhez hasonló felépítéssel. A felhasználót a játék folytatásáról kérdezi.
4. save_topscore()
    - A játékost visszajelzését várja az elért pontszám dicsőséglistára való mentése kapcsán.
5. menu(buttons, data=None)

- A fenti függvények által is használt menü mechanizmus, aminek fő eleme egy `while` loop, ami választásig/kilépésig fut, akkor pedig a választott értékkel (kilépésnél None) tér vissza.
- Ez írja ki a gombokat, és érzékeli kezeli azok kiválasztását is, billentyűkezeléssel.

6. select(func, data=None)
   - A fenti *menu()* hívja meg ENTER billentyű hatására, és az adott gombhoz rendelt funkció alapján visszatér a megfelelő függvényhívással. A legtöbb funkcionalitás a függvényen belül van lekezelve, néhány esetben viszont (amikor az eredendő függvényhívás helyén a lokális változók miatt szükséges) csak a funkció nevével tér vissza a függvény, és a választás a *select()*-ten kívül lesz lekezelve.
7. setting_adjust(setting, label, min_val, max_val, delta=1)
   - A beállítások menüben használt függvény, ami az adott beállítás módosítását teszi lehetővé, a fel-le, ESCAPE/ENTER billentyűk segítségével.
   - ESCAPE esetén a beállítás visszaáll a kezdő értékre, a változtatás nem kerül mentésre, ENTER-nél viszont igen, a függvény pedig mindkét esetben az új beállítással tér vissza.
   - Az érték a paramétereknek megfelelően állítható, a *delta*-val az állíthatóság mértékét, a *setting*-gel a konkrét beállítást, a *label*-lel a kiírandó szöveget, a *min*- és *max_val*-lal pedig az állíthatóság intervallumát szükséges megadni.
8. options(pos, settings=None)
   - A fenti menükhöz hasonló beállítások menü, ami (ha még nincs), létrehoz egy *Options* objektumot is.
   - Ha a felhasználó semmit sem állít be, a *main_menu()*-be tér vissza, egyébként pedig önmagát hívja meg újra, az értékek beállítása után, az új beállításokkal.
9. get_scores()
   - A `highscores.txt`-ből beolvasott pontlistával tér vissza, ami *Score* objektumokból áll.
   - Ha a file nem található, a visszatérési érték None, ha pedig a pontszám nem alakítható integerré, -1-gyel.
10. list_scores(scorelist, pos, data=None)
    - Kilistázza a paraméterként átadott toplista bejegyzéseit a megadott pozícióra, számozva (és top 3 esetén színezve is) azokat.
    - ESCAPE lenyomására visszatér a *main_menu()* hívásával.

- A hibás/hiányzó file-okat a függvény a felhasználó tájékoztatásával kezeli le.

11. add_score(score)
    - A felhasználót egy név megadására kéri -amennyiben a felhasználó el akarja menteni a pontszámát- és visszatér a nevet és új pontszámot tartalmazó pontlistával, egyébként pedig None-nal.
    - Az új pontszám első körben a pontlista végére kerül, az pontszámok alapján rendezve lesz, és ha így a lista több, mint 5 elemből áll, az utolsó elem törlődik.
    - Ha beolvasott pontlista hibás, jelzi ezt a felhasználónak.
12. write_score(scorelist)
    - Elemenként kiírja a paraméterként megadott pontlistát `név: pontszám` formátumban, a `highscores.txt` file-ba.

control.py

1. rotate(tetro)
    - Egy új tetromínót generál a megadottnak megfelelően, majd óramutató járása szerint 90°-kal elforgatja, és ezt adja vissza.
2. make_random(pos)
    - A hét lehetséges betű (I, J, L, O, S, T, Z) valamelyikét választja véletlenszerűen, és visszatér a paraméterként megadott pozícióra, a választott betűvel generált tetromínóval.
3. ingame(tetro, field)
    - A játék közbeni irányításért felelős függvény, aminek visszatérési értéke egy tuple, ami azt jelzi, hogy (0.) a játék folyamatban van-e, és (1.) hány pluszpontot kell hozzáadni a játékos összpontszámához.
4. post_move(tetro, dir)
    - A paraméterként megadott tetromínó szerint létrehoz egy újat, amit a *dir* paraméternek megfelelően mozgat, és visszaadja a mozgatott tetromínót.
5. move_valid(tetro, field)
    - Eldönti, hogy a paraméterként adott tetromínó minden egysége a játéktéren belülre esik-e, és ezzel a logikai értékkel tér vissza.

> Legtöbbször a post_move()-val együtt használatos.

6. rightmost(tetro)
   - A tetromínó legjobboldalibb egységének relatív X pozícióját adja vissza, a bal felső egységhez képest.

   > Ez forgatásnál használatos.

7. hit(tetro, field)
   - Eldönti, hogy a megadott tetromínó által elfoglalt cellák már foglaltak-e, tehát hogy van-e ütközés már eltárolt elemmel.

8. make_field(fsize)
   - Létrehozza és visszaadja a pálya tárolására használt 2 dimenziós listát (sor:oszlop 2:1 arányban), kezdetben 0-ákkal feltöltve, méretét az *fsize* adja meg.

9. update_field(tetro, field)
   - Visszaadja a paraméterként megadott tetromínó megfelelő pozícióba való eltárolása után keletkezett *field* listát.
   - Az elfoglalt cellákat a tetromínó alakját reprezentáló betűvel írja felül.

10. store_regen(tetro, field, next)
    - A fenti függvény hívásával eltárolja a paraméterként adott tetromínót a pályába, az aktuális tetromínót az eddig következőre állítja, majd következőnek létrehoz egy véletlenszerűen generáltat, és visszaadja azt.

11. line_full(field)
    - Ellenőrzi, hogy az adott pályán van-e teli sor (amiben nem szerepel 0), és visszatér a megfelelő logikai értékkel.

12. delete_full(field)
    - Törli a teli sorokat a pályáról, felülírja a cellák tartalmát 0-val.
    - Törlésenként a feljebb lévő sorok elemeit mindig 1-gyel lejjebb mozgatja, amíg szükséges.
    - A teli sorokért járó plusz pontszámmal tér vissza.

13. speed_sec(level)

- Visszatér az adott szinthez tartozó idővel, amit a program az ütközések és az esés ütemezéséhez használ.

14. save_game(tetro, field, next, points, level)
    - Elmenti a paraméterként adott adatokat a `save.txt` file-ba, felülírva a file korábbi tartalmát.
15. load_game(file)
    - Beolvassa a paraméterként megadott file-t, és felépíti belőle a játékban használt adatszerkezetet.
    - Ha a file nem található, None-nal tér vissza.

draw.py

1. ground(field)
    - Kirajzolja a pályát (jelenlegi állapotában) a terminálra, aminek határait és tartalmát karakterek jelzik.
2. nextsection(field, next)
    - A soron következő tetromínó jelzésére használt dobozt írja ki a terminálra, illetve magát a tetromínót is, a dobozon belülre.
3. valsection(field, val, posy, label)
    - Paraméterként megadható adatokkal rajzol dobozt, 1 egység magassággal, aminek szélessége igazodik a benne megjelenített adatéhoz.
4. screen(tetro, field, next, points, level)
    - A játékon belüli képernyőn rajzolja a terminálra, tehát a pályát, a pályaméret-, pontszám-, és szint szekciókat, és a jelenleg irányítható (aktív) tetromínót.
5. get_color(shape)
    - Visszaadja a tetromínóhoz tartozó színt, a pyconio modul színei közül, a *shape* paraméterként megadott betű alapján.
6. Tetromino.print(self)
    - Kirajzolja a tetromínót a terminálra.

- Mivel a tetromínó sztringgé alakításánál újsor karakter is keletkezhet, kiíráskor újsor karakternél vissza kell tenni a kurzort a megfelelő pozíciókra, nem pedig elkezdeni a kiírást az x=0-ás pozíciótól.

  > Mivel a fenti függvényt minden esetben Tetromínókhoz használjuk, classon belül lett definiálva, így nem paraméterként kell megadni azt minden esetben, hanem annak egy metódusaként hivatkozunk rá.
  > Így talán többet mond a függvény külső szemlélőnek.

7. cursor(show)

   - A terminálon belüli kurzor elrejtésére (vagy kijelzésére, a logikai *show* paraméternek megfelelően) szolgáló ANSI escape sequence-t ír ki a terminálra, amennyiben a futtató platform nem Windows (a Windows console nem támogatja az ANSI escape sequence-eket).

8. logo(file="logo.txt")

   - A logos/ mappában lévő megadott *file* (alapértelmezetten logo.txt) tartalmát írja ki a terminálra, különböző színekkel. A *file*-ban a különböző színnel kirajzolandó szekciók `|` karakterrel vannak jelölve.