TETRIS

Fejlesztői dokumentáció

Környezet

A játék a python sztenderd könyvtárat, illetve megjelenítéshez a pygame modult használja, utóbbit először installálni kell:

```
> py -m pip install -U pygame --user
```

(Az installáció módja az op. rendszertől és a meglévő python verziótól függően eltérő lehet, több információ erről: https://www.pygame.org/wiki/GettingStarted)

A játék python 3.7.0-ben íródott és lett letesztelve.

Projekt felépítése

`menu.py`
Fő modul, ez a fájl a program belépési pontja. Megjeleníti a főmenüt, és meghívja a program többi részét a felhasználó kérésének megfelelően.

`game.py`
Ez a fájl tartalmaz mindent, ami a konkrét játékhoz tartozik, a mögöttes logikai rendszert, és ami a megjelenítéshez szükséges.

`leaderboard.py`
Tartalmaz mindent, ami a pontszámok fájlba mentéséhez, és fájlból kiolvasásához szükséges, képes megjeleníteni az elmentett eredményeket.

`utility.py`
Ez a fájl tartalmazza az olyan kódrészeket, amelyeket több másik fájl is használ, pl. színek, betűtípusok stb.

`fonts` mappa
A felhasznált betűtípusokat tárolja.

Adatszerkezetek

A program alapvető felépítése:
A `~Screen` nevű osztályok a megjelenítést teszik lehetővé, a `~Model` nevű osztályok pedig a megjelenítéshez szükséges technikai hátteret biztosítják. Egyszerűbb ablakok megjelenítésénél nem társul `~Model` osztály a `~Screen` osztályhoz. Ahol van `~Model` ott a `~Screen` osztályt a `~Model` egy példányával kell inicializálni.

`game.py`:

States
Azt a háromfajta állapotot tárolja, ami jellemezhet egy cellát:
EMPTY – üres
FILLED – teli, le van rakva,
MOVING – teli, még mozog

Cell
Cella típus, van színe és állapota.

Coordinate
Koordináta típus, van x és y koordinátája.

Tetromino
Tetrominónak hívnak egy hulló elemet a játékban, ez az osztály egy ilyen elemet definiál.
Tároljuk:
- alakját, a beillesztés helyéhez képest relatív koordináták listájaként,
- forgáspontját, ugyanígy relatív koordinátaként,
- színét.

GameModel
Ebben az osztályba tároljuk mindazt, ami szükséges a játék működéséhez.
Tároljuk:
- a 7 fajta alakú tetrominót a `TETROMINOS` listában, ezek Tetromino típusúak
- pálya szélességét,
- pálya magasságát,
- az ezek alapján létrehozott pályát, ami egy mátrix, amilyen magas, annyi sora, amilyen széles, annyi oszlopa van, az elemei cellák. Ebben tároljuk el a pálya aktuális állását, és ennek a segítségével jelenítjük meg azt.
- a játék során elért pontokat, ez induláskor 0
- elvesztett-e a játék, True/False
- az éppen mozgó elem tengelyének helyét, ez induláskor None, később egy Coordinate objektum
- mi lesz a következő tetromino, kezdetben egy véletlenszerű eleme a `TETROMINOS` listának

GameScreen
Tárolja a játék megjelenítéséhez szükséges konstansokat (cellaszélesség, pálya oldalán lévő mezők méretei), és hogy szüneteltetve van-e a játék.

SettingsScreen
Tárolja a pálya magasságát és szélességét, amelyek értéke alapértelmezetten 20 és 10, de eltárolja, ha ezeken változtattunk.

`leaderboard.py:`

`Record`
Egy elért eredmény típusa, eltárol egy nevet és egy pontszámot.

`LeaderboardModel`
Tárolja:
- a fájl nevét, ahova írja és ahonnan kiolvassa a ranglistát,
- az elért eredményeket, egy pontszám szerint csökkenő sorrendbe rendezett listában, ezek Record típusúak

`NameEntryScreen`
Tárol egy nevet, amit a program akkor kér be a felhasználótól, amikor a pontszámát el akarja menteni.

`utility.py:`

`Colors`
Tárolja a megjelenítéskor használt színeket, RGB színkódok formájában.

`Fonts`
Tárolja a megjelenítéskor használt betűtípusokat.

`Actions`
Megmondja, mit kell csinálni azután, hogy a jelenlegi képernyő visszatért:
QUIT - Ki kell lépni az alkalmazásból
OK - Tovább lehet menni
CANCEL - Meg lett szakítva a művelet

Függvények

A `~Screen` osztályokban lévő `show()` függvények ugyanazon pygame adta logika szerint vannak felépítve:
Inicializáljuk az adott ablakot, majd belépünk a főciklusba, melynek egy futása egy képkocka. Minden egyes képkocka kirajzolása úgy történik, hogy először feldolgozzuk az eseményeket (billentyűzet, időzítő, kilépés), majd kirajzoljuk a képernyőre a képkockát, frissítjük az ablakot, és várunk a következő képkockáig. Másodpercenként 60 képkockát jelenítünk meg.
A `main()` függvény is így működik, azzal a különbséggel, hogy az a pygame inicializációját is elvégzi az elején.

`game.py:`

`next_tetromino()`
Visszatérési értéke a következő beillesztendő tetromino, és módosítja az ezután következő tetromino értékét egy a TETROMINOS listából vett random tetrominora.

`insert()`
Berak egy új tetrominót a pályára. Megadja, melyik cellája a mátrixnak a tengely, és beilleszti a tetromino relatív koordinátáit a mátrix abszolút koordinátarendszerébe, továbbá beállítja a beillesztett cellák állapotát mozgásban lévőre.

`tick()`
Megvizsgálja minden cellájára a lefele mozgó tetrominonak, hogy mehet-e lejjebb (van-e alatta másik teli cella, vagy nem fogy-e el a játéktér). Ha mehet lejjebb, akkor alulról haladva cellánként belerakja magát az eggyel lentebbi cellába, saját helyét pedig kitörli. Ha nem mehet lejjebb, akkor a játéktér fix elemévé teszi a celláit, ha lett ezen elem lerakásával teli sor, akkor azt kitörli és lejjebb ugratja a táblát, megvizsgálja, nem ért-e véget a játék (azaz léphet-e be még a játéktérre új elem). Ha nem ért véget, akkor bedob egy új elemet, ha véget ért, akkor ezt az információt eltárolja, és nem dob be innentől új elemet.

`move_left()`
Megvizsgálja minden cellájára a lefele mozgó tetrominonak, hogy mehet-e balra (van-e mellette másik teli cella, vagy nem fogy-e el a játéktér). Ha mehet balra, akkor balról haladva cellnáként belerakja magát az eggyel balra lévő cellába, saját helyét pedig kitörli.

`move_right()`
Megvizsgálja minden cellájára a lefele mozgó tetrominonak, hogy mehet-e jobbra (van-e mellette másik teli cella, vagy nem fogy-e el a játéktér). Ha mehet jobbra, akkor jobbról haladva cellánként belerakja magát az eggyel jobbra lévő cellába, saját helyét pedig kitörli.

`rotate()`
Elforgatja az aktuális tetrominót 90 fokkal az óramutató járásával ellentétes irányba.
Létrehoz egy üres mátrixot, amibe belemásolja a pálya mátrix teli celláit, majd a mozgó celláit. A mozgóknál először kiszámítja egy cella új, elforgatott koordinátáit, és ha beilleszthető az új mátrixba (nem lógna ki a pályáról és nem írna felül már teli cellát), akkor beilleszti. Ha akár egy cella beillesztése is akadályba ütközne, visszatér a függvény a forgatás végrehajtása nélkül. Ha minden cella beillesztése sikerült, akkor kicseréli a régi pályamátrixot az új mátrixra.

`line_clear()`
Törli a teli sorokat. Alulról vizsgálja soronként a játéktábla mátrixot, ha talál egy sort, ahol minden cella teli, akkor mindent, ami afelett van, egy sorral lejjebb hoz. Ha törlés volt egy adott sorban, akkor azt ismételten addig ellenőrzi, amíg biztos, hogy nem teli, aztán folytatja felfele a keresést.

`check_end_condition()`
Végignézi a játéktér mátrix felső 3 sorát (a 2 legfelső sor nincs megjelenítve, csak az új elemek betöltésére van fenntartva), és ha nem üres az összes cella bennük (azaz már telítődött a valós játéktér), akkor visszaadja, hogy véget ért a játék, ellenkező esetben pedig, hogy nincs még vége a játéknak.

`draw()`
Kirajzolja a játékteret, a következő három függvény segítségével.

`draw_field()`
Kirajzolja magát a pályát a pályamátrix alapján - a felső két sort nem rajzolja ki, ide töltődnek be az új tetrominók.

`draw_preview()`
Kirajzolja a következő elemet megjelenítő mezőt.

`draw_scoreboard()`
Kirajzolja a pontszámot megjelenítő mezőt.

`leaderboard.py:`

`record_score()`
Paraméterként megkap egy pontszámot és egy nevet, megnyitja/létrehozza a fájlt, ahova eltárolja ezeket. Tehát a fájlban egy eredmény egy sor, a következő formátumban: Név<tab>Pontszám.
Ezután létrehoz egy Record típusú objektumot ezen két paraméter alapján, és ezt az új eredményt berakja az eredménytároló listába, ami már pontszám szerint csökkenő sorrendbe van rendezve, így csak a helyét kell megkeresnie. Ha belerakta, visszatér. Ha a hely megtalálása nem sikerült – mert a lista üres, vagy mert az utolsó helyre kell beilleszteni – akkor a végén hozzáfűzi az elemet a lista végéhez.

`read_records()`
Beolvassa fájlból az eltárolt eredményeket, és visszatér egy pontszám szerint csökkenő sorrendbe rendezett eredményeket tároló listával.
Először létrehoz egy üres listát, majd megnézi, létezik-e az adott fájl, ahonnan be kell olvasnia az eredményeket, ha igen, akkor soronként végig haladva a fájlon hozzáadja az üres listához az eredményeket Record típusú objektumként (1 sor = 1 eredmény). Ezután rendezi a listát pontszámok szerint csökkenő sorrendbe, és visszaadja ezt.
Ha nem létezik az adott fájl, visszaadja az üres listát.