Orbán Dávid bemutatja:

**Slitherlink**

Tartalom

[Bevezető: 2](#_Toc197425954)

[Technikai adatok: 2](#_Toc197425955)

[A keret 3](#_Toc197425956)

[A program felépítése 4](#_Toc197425957)

[A függvények: 6](#_Toc197425958)

# Bevezető:

A Slitherlinket azért választottam, mivel érdekes kihívásnak tűnt, hogy megírjak egy olyan programot, amely képes egy NP nehéz probléma megoldására tűrhető időn belül.

## Technikai adatok:

A projektre a Python-t választottam, mivel annak ellenére, hogy nem volt személyesen még vele sok tapasztalatom, tudtam, hogy a flexibilitása miatt sokkal jobban illik ide, mint a C++, amit leggyakrabban használok.

Maga a csatolt mappa két mappából és két fájlból áll: a *pregenboards* előre regenerált játéktáblákat tárol, a *resources* a program által használt betűtípusokat illetve konfigurációs fájlokat tartalmaz, a *main.py* magát a programot, a *requirements.txt* pedig a használt könyvtárakat, illetve verzióikat, a pip által generált formátumban. (A *pip install -r 'requirements.txt'* parancs megkönnyíti az importálásukat.)

(Kérdéses a relevanciája, de azért nem hagyom ki: a saját számítógépemen Visual Studio Code-ban dolgoztam egy Python által létrehozott virtual environment-ben: a Python installációm verziója 3.13.3, a pip-é 25.1.1; továbbá amennyiben valamilyen esetleges probléma merülne fel a megadott fájlokkal, a [projekt Github-on is elérhető](https://github.com/odbv/Slitherlink-Termtud).)

# A keret

Elsősorban essen szó a program keretéről, vagyis minden elemről amely nem a megoldásért, vagy egy új tábla generálásáért felel.

A program négy fő globális változón alapul: az *n* és *m* a jelenleg betöltött tábla magasságát, illetve szélességét tárolják el, a *v* a játékos által is modosítható táblázatot, míg a *sol* a megoldást tárolja. Ez a két mátrix a következőképpen képezi le a Slitherlink táblát: az n \* m-es látható táblázatot mint (2 \* n + 1) \* (2 \* m + 1) táblázat tárolja el. A látható értékek a dupla páratlan indexű cellákban vannak, a dupla páros indexű cellákban a pontok (amik nem befolyásolják a táblázatot, de majd később hasznosak lesznek), a többi cellában pedig az élek.

A black arrow pointing to a grid

AI-generated content may be incorrect.

(Egy egyszerűbb feladvány és a vizualizálása annak, hogy a megoldásának melyik elemét a táblázat melyik cellája tárolja.)

## A program felépítése

A következő ábra demonstrálja a program felépítését a függvények nézőpontjából:

A diagram of a software system

AI-generated content may be incorrect.

Ahol a dupla-irányítású nyíl azt jelenti, hogy a függvény futása után a meghívója folytatja a saját feladatát, az egyirányú nyíl pedig azt jelzi, hogy a meghívott függvény meghívása után az adott függvénynek nincsenek további feladatai.

A program a következőféleképpen működik:

* a main függvény meghívja az initboards-ot, amely detektálja a mappát, ahol a file található, illetve a getboard-ot, amely betölti a v-be az egyik előre kiválasztott előre generált táblát (egyeseknek közülük a megoldása is előre le van generálva, abban az esetben azt is betölti)
* a main függvény ezután meghívja az initwindow függvényt, ahol a játék maga fut: a pygame megrajzolja a táblát, érzékeli a klikkeket, et cetera. Az initwindow öt függvényt képes meghívni:
  + A checkifvalid leellenőrzi, hogy a mostani táblakonfiguráció egy helyes megoldás-e. Ez által a játék manuálisan is játszható.
  + A calculatesolution kiszámolja a megoldást, amennyiben az szükséges. (Például akkor nem muszáj, hogyha előre generált tábla aminek meg volt jegyezve a megoldása, vagy ha általunk generált tábla.)
  + A három newgame funkció mind egy-egy új játék felállításáért felel:
    - A newgame\_pregen kinyitja a *pregenboards* mappát, ahonnan a felhasználó kiválaszthatja, hogy melyik táblát szeretne betölteni (amely a getboard funkció segítségével történik)
    - A newgame\_genboard bekér két értéket, az új-n-et és az új-m-et, és ezeket továbbadja a genboard függvénynek, amelyről majd bővebben a későbbiekben lesz szó.
    - A newgame\_insertboard bekér egy Loopy stílusú ID-t, és az alapján betölt egy új táblát. (Nem muszáj pont azt a struktúrát tartani: igazából az mxn és a kettőspont között akármi lehet, mert azt figyelmen kívül hagyja, így egyes más implementációkkal is kompatibilis.)

## A fontosabb függvények:

A keret két legfontosabb függvénye az initwindow illetve a checkifvalid. A következőkben ennek a kettőnek a működéséről lesz szó.

### Initwindow

A grafikus felület kezeléséért a Pygame (Community Edition) illetve a Pygame GUI nevű könyvtárak felelnek a legnagyobb mértékben.

Mielőtt elkezdene futni maga a game-loop, a háttérre odarajzolja a pontokat és számokat, illetve kiszámolja az összes él-téglalap adatait. A kezdőgombokat is ekkor rajzolja a képernyő jobb oldalára. Magukat az éleket minden ciklusban újrarajzoljuk, a következőféleképpen: amennyiben egy él nem aktív, rajzolunk egy dezaktivált élet, és azon keresztül detektáljuk, hogy az egér arra a pozicióra klikkelt-e, amelyik esetben átállítjuk aktív éllé.

Párhuzamosan történik a gombok rendezése is: összesen 4 konfiguráció van jelen:

* A kezdő, amikor jelen vannak a *New Game*, *Check if valid* és *Show Solution* (vagy *Hide Solution*) gombok
* A *New Game* megnyomása után ezeket lecseréli a három *newgame* funkciónak megfelelő gomb: a *Load board*, *Generate board*, illetve az *Insert board*