**The Executors: Hangman game - Akasztófa**

1 . Definiáljunk egy play nevű metódust, amely két paramétert vár: word és lives.

2. A word paraméterben adjuk át a play metódusnak a megtippelt szavakat és a lives paraméterben adjuk át a maximális hiba számot (nehézségi szinttől mérten)

3. A kezdő állapot úgy néz ki, hogy az éppen aktuális kitalálni való szó betűit underscore-ral helyettesítve jelenítjük meg.

4. Bekérünk a játékostól egy betűt és ha olyan betűt tippel, ami szerepel a szóban, akkor az adott helyen lévő underscore-t a megfelelő betűvel helyettesítve jelenítjük meg.

5. Amikor a beírt betű nem szerepel a szóban, akkor a játékos veszít egy életet, azaz, elkezdjük megrajzolni az emberkét.

6. Ha egy olyan betűt ír be a játékos, amit már beírt korábban (tehát mindegy, hogy szerepelt-e a szóban vagy sem), akkor nem történik semmi, nem vesz le életet sem.

7. Legyenek a képernyőn kilistázva azok a betűk, amiket már beírt, és rosszak voltak attól függetlenül, hogy ezt korábban már beírta-e.



8. A játékos akkor nyer, amikor az összes betű megjelenik a szóból, tehát nincs több underscore.

9. A játékos akkor veszíti el a játékot, ha életszám == rossz tipp számmal (akkor nem számoljuk a rossz tippet(nem vonunk le életet), amikor az adott megtippelt betű már egyszer volt.)

10. Amikor a játékos beüti, hogy ‘quit’, akkor kiírjuk, hogy ‘good-bye’ és befejeződik a játék. “If you want to quit, type “quit”! If input == “quit”: print(“Goodbye!), end the game.

11. A játékos nagybetűt és kisbetűt is bevihet, nem különböztetjük meg: h == H.

Viszont a képernyőn a feladványban megkülönböztetve jelenítjük meg: Paris, Salvador Dali.

12. A kisbetűt és a nagybetűt is elfogadjuk a játékostól.

13. Ha egy feladványban van kis h és nagy H, a játékos beüt kis h-t vagy nagy H-t, mindkét esetben a kis és nagy H is jelenjen meg: H\_ \_ \_ h \_ \_ \_h \_ \_.

14. Amikor az input ismétlődését vizsgáljuk, akkor a h = H, tehát ha először h-t ütött be, utána pedig H-t, akkor a második már ismétlődésnek számít.

15. A képernyőn úgy jelenítjük meg a betűket, ahogy az normálisan is megjelenne, tehát ha tulajdonnév a feladvány, akkor az első betűjét nagy betűként jelenítjük meg.

!!! A feladványban szereplő szó 0. indexe mindig nagybetű letter.capitalize()

16. Megjelenítés - grafika ASCII art-tal jelenítjük meg a játékot terminál ablakban.

17. Az élet számát egy ASCII art-ban megjelenített akasztófára rajzolt emberke mutatja.

18. Az akasztófa megjelenítése szoros összefüggésben áll a lives paraméterrel (ami minimum 3 és 7 között mozog), tehát a végeredmény ugyanaz az easy és a hard módban is, de az easy módban lassabban/több lépésben rajzolja meg ugyanazt

19. Van egy előre megalkotott szó lista (feladvány lista) amiből a szavakat betölti, kiválasztja.

Nem így kell: myList=[Paris, London, Prague]

20. A játék random választ ebből a listából minden egyes feladványnál.

(set - variable type: nem lehet duplikáció és random köp??)

random.choice() built in func ??

21. A játék használ egy txt fájlt és abból random választ ki egy feladványt. (countries-and-capitals.txt)

**22. Különböző nehézségi szint -** A játékot különböző szinteken lehet játszani.

23. A játék azzal indul, hogy a játékos választ egy nehézségi szintet, amin játszani akar.

Ez nem befolyásolhatja a play() funciont. Tehát ezt nem a play() function-ön belül implementáljuk, hanem azon kívül.

24. easy: Paris, London, Budapest - 7 élet: fej, törzs kettőből, bal kéz, jobb kéz, bal láb, jobb láb

hard: Nagytarcsa, Taktaharkany, nehéz város nevek.. - 4 élet fej, törzs, kezek, lábak

25. A játék állapota == megjelenített és a tévesen megtippelt betűk, ezeket ne string típusban tárolja, mert az megváltoztathatatlan, hanem változtatható típusú változóban pl list vagy set

26. Set típust használj olyan változók tárolására, amikor nem lehet duplikációd a listában.

27. Legyen legalább 3-6 saját funcion definiálva, amely valamelyest független az üzleti logikától! Pl ezekre létrehozni: A betűbevitelre, a megjelenítés részeire, vagy a menü (nehézségi szint) megjelenítésére.

English Instructions:

1. Implement the play(word, lives) function as a basic hangman game.

2 .The function uses its parameter word as the word to guess and lives to as the number of available mistakes

3 The initial game state is displayed as \_ \_ \_ \_ \_ \_ \_ \_ (one underscore for each letter in word)

4. The game state is displayed as \_ o d \_ \_ o o \_ if letters 'd' and 'o' have been revealed. It is possible to make guesses, and letters that occur in the word are revealed

5. When a guessed letter does not occur in word, the player loses one life

6. When a guess is repeated (regardless of its occurrences), the player is notified, and nothing happens

7. When a guess is wrong (either a new or a repeated letter), the already tried missing letters are shown to the user

8. The player wins when all the letters in word have been revealed

9. The player loses when misses a letter for the livesth time (not counting repeated guesses)

10. When the player types 'quit' as input, the program says good-bye and terminates

**11. Case sensitivity -** The game play is case insensitive while the word display is case sensitive

12. Both uppercase and lowercase letters are considered as valid input

13. Uppercase and lowercase letters guesses reveal the same letters (e.g. both c and C guesses reveal all the cs in the word, regardless of their case)

14. Letters of different cases behave as if they were the same when checking repetitions (e.g. entering c after a C would count as a repetition)

15. On the displaying side, however, letters are revealed as they originally appear in word (e.g. successfully guessing c shows C \_ \_ \_ c \_ \_ \_ for Codecool)

**16. Graphics -** Add ASCII art to visualize lives left.

17. The game state display is accompanied by an ASCII art depending on the lives left

18. The art sequence is adapted to the starting value of the lives parameter (at least between 3 and 7) - this means that the loosing picture is always the same

**19. Load words - Feladványok betöltése -** The game uses a random word from a pre-defined word collection.

20. The game randomly picks a word at each run

21. The game randomly picks a country from countries-and-capitals.txt

**22. Different levels -** The program allows to play the game on different levels.

23. The game starts with a menu for picking a difficulty level. You should not change the play() function, though!

24. The word-pool and the number of lives depend on the chosen level

25. As strings are immutable (i.e. you cannot change its letters) it is a better idea to store the state of the game (like the revealed and missed letters) with the help of mutable structures (like lists or sets)

26. You are advised to use a set data structure when you have a collection which cannot have duplicated elements

27. Try to create a few (3-6) functions for features that are somewhat separated from the main process (e.g. dealing with the inputs, parts of the display, or the menu). Think of the input requirements and the results of these units! Add the necessary inputs as parameters, and return the results that is needed by the caller side!

Menu-re példa:



