**Hangman**

Implement the popular Hangman game. Add a full fledged game flow with a main menu and optionally some cool graphics in the console!

## What are you going to learn?

* Use lists and strings.
* Handle files.
* Use and validate user input.
* Handle normal and edge cases.
* Create ASCII art.

## Tasks

**Hangman game**

Implement a basic hangman game flow. You can use any (even a constant) word to guess, focus on the guessing logic and the termination of the game.

1. The game requires the user to guess all the letters belonging to a word.
2. The initial game state is displayed as \_ \_ \_ \_ \_ \_ \_ \_ (one underscore for each letter in word).
3. The game state is displayed as \_ o d \_ \_ o o \_ if letters 'd' and 'o' are revealed.
4. 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 previous wrong letters are shown to the user.
8. The player wins when all letters in word are revealed.
9. The player loses when the number of wrong guesses is higher than the initial value of lives parameter (not counting repeated guesses).
10. When the player types 'quit' as input, the program says good-bye and terminates.

**Case sensitivity**

The gameplay is case insensitive, the word display is case sensitive.

1. Both uppercase and lowercase letters are considered valid input.
2. Uppercase and lowercase letter guesses reveal the same letters (e.g. both c and C guesses reveal all the cs in the word, regardless of their case).
3. 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).
4. On the displaying side, however, letters are revealed as they originally appear in word (e.g. successfully guessing c shows C \_ \_ \_ c \_ \_ \_ ).

**Graphics**

Add ASCII art to visualize lives left.

1. The game state display is accompanied by an ASCII art depending on the number of lives left.
2. The art sequence is adapted to the starting value of the lives parameter (at least between 3 and 7) – this means that the game over art is always the same.

**Load words**

The game uses a random word from a pre-defined word collection.

1. The game randomly picks a word at each run.
2. The game randomly picks a country from countries-and-capitals.txt.

**Different levels**

The program allows the user to play on different levels.

1. The game asks the user to pick a difficulty before starting
2. The word-pool and the number of lives depend on the chosen level.

## Hints

* Store the state of the game (such as the revealed and missed letters) with the help of mutable structures (such as lists or sets).
* Use a set data structure when you have a collection that cannot have duplicate elements.
* Try to create a few (3-6) functions for features that are somewhat separated from the main process (such as 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!
* Ideal team size is 2. Maximum team size is 3.

Wisielec Wdrożenie popularnej gry w wisielca. Dodaj pełnoprawny przepływ gry z menu głównym i opcjonalnie fajną grafiką w konsoli! Czego się nauczysz?

• Używać list i ciągów.

• Obsługiwać pliki.

• Używać i weryfikować dane wejściowe użytkownika.

• Obsługiwać przypadki normalne i skrajne.

• Tworzyć grafikę ASCII.

Zadania Gra w wisielca Wdrożenie podstawowego przepływu gry w wisielca. Możesz użyć dowolnego (nawet stałego) słowa do odgadnięcia, skup się na logice zgadywania i zakończeniu gry.

1. Gra wymaga od użytkownika odgadnięcia wszystkich liter należących do słowa.

2. Początkowy stan gry jest wyświetlany jako \_ \_ \_ \_ \_ \_ \_ \_ (jeden znak podkreślenia dla każdej litery w słowie).

3. Stan gry jest wyświetlany jako \_ o d \_ \_ o o \_, jeśli litery „d” i „o” są ujawnione.

4. Można zgadywać, a litery występujące w słowie są ujawniane.

5. Gdy odgadnięta litera nie występuje w słowie, gracz traci jedno życie.

6. Gdy odgadnięcie jest powtarzane (niezależnie od jego występowania), gracz jest powiadamiany i nic się nie dzieje.

7. Gdy odgadnięcie jest błędne (nowa lub powtórzona litera), użytkownikowi pokazywane są poprzednie błędne litery.

8. Gracz wygrywa, gdy wszystkie litery w słowie zostaną ujawnione.

9. Gracz przegrywa, gdy liczba błędnych odgadnięć jest wyższa niż początkowa wartość parametru żyć (nie licząc powtórzonych odgadnięć).

10. Gdy gracz wpisze „quit” jako dane wejściowe, program żegna się i kończy działanie.

Rozróżnianie wielkości liter

Rozgrywka nie uwzględnia wielkości liter, wyświetlanie słów uwzględnia wielkość liter.

1. Zarówno wielkie, jak i małe litery są uważane za prawidłowe dane wejściowe.

2. Odgadywanie wielkich i małych liter ujawnia te same litery (np. zarówno odgadywanie c, jak i C ujawnia wszystkie c w słowie, niezależnie od wielkości liter).

3. Litery w różnych przypadkach zachowują się tak, jakby były takie same podczas sprawdzania powtórzeń (np. wpisanie c po C byłoby traktowane jako powtórzenie).

4. Po stronie wyświetlania litery są jednak ujawniane tak, jak pierwotnie pojawiły się w słowie (np. pomyślne odgadnięcie c pokazuje C \_ \_ \_ c \_ \_ \_ ). Grafika Dodaj grafikę ASCII, aby zwizualizować pozostałe życia. 1. Wyświetlanie stanu gry jest połączone z grafiką ASCII w zależności od liczby pozostałych żyć. 2. Sekwencja grafik jest dostosowana do wartości początkowej parametru żyć (przynajmniej od 3 do 7) – oznacza to, że grafika kończąca grę jest zawsze taka sama. Wczytaj słowa Gra wykorzystuje losowe słowo z predefiniowanej kolekcji słów.

1. Gra losowo wybiera słowo przy każdym przebiegu.

2. Gra losowo wybiera kraj z pliku countries-and-capitals.txt.

Różne poziomy Program umożliwia użytkownikowi grę na różnych poziomach.

1. 1. Gra prosi użytkownika o wybranie poziomu trudności przed rozpoczęciem
2. 2. Pula słów i liczba żyć zależą od wybranego poziomu. Wskazówki
3. • Przechowuj stan gry (taki jak odkryte i pominięte litery) za pomocą zmiennych struktur (takich jak listy lub zestawy).
4. • Użyj struktury danych zestawu, gdy masz kolekcję, która nie może mieć duplikatów elementów.
5. • Spróbuj utworzyć kilka (3-6) funkcji dla funkcji, które są w pewnym stopniu oddzielone od głównego procesu (takich jak obsługa danych wejściowych, części wyświetlacza lub menu). Pomyśl o wymaganiach wejściowych i wynikach tych jednostek! Dodaj niezbędne dane wejściowe jako parametry i zwróć wyniki, które są potrzebne stronie wywołującej!
6. • Idealny rozmiar zespołu to 2. Maksymalny rozmiar zespołu to 3.