Gra wisielec

Autor: Paweł Wiśniewski

Gra w postaci skryptu powłoki bash. Projekt powinien zawierać słowa pobierane ze słownika znajdującego się w katalogu /usr/share/dict/. Ze wględu na używaną przeze mnie dystrybujcje Fedora Workstation, miałem do dyspozycji tylko słownik angielski. Znalazłem w sieci plik z polskimi słowami do gier i wykorzystałem go jako zamiennik. Wisielec powinien być generowany z każdą źle odgadniętą literą. Pierwotnie hasło jest ukryte pod postacią “\_ \_ \_ \_”, a litery odgadnięte są podmieniane za znak “\_” np. “\_ \_ a \_ \_a”. Kiedy gracz odgadnie wszystkie litery wygrywa. Kiedy ilość błędów jest zbyt duża wisielec rysuję się do końca i gracz przegrywa.

Skrypt rozpocząłem od utworzenia pliku hangman.bash i skopiowania pliku ze słowami do tego samego katalogu. W skrypcie od razu utworzyłem zmienną przechowującą ścieżkę do pliku. W tym samym momencie wpadłem na pomysł wyszukania słowa, które będzie hasłem w grze. Polegało to na tym, że słowa w pliku są zapisane każde w osobnej lini. Mogłem więc sprawdzić ilość lini w pliku, następnie wylosować sobie liczbę w tym przedziale i wyciągnąć dany wiersz(słowo) znajdujący się pod tym numerem. Tutaj napotkałem problem gdyż polecenie “wc -l" zwracało ilość lini i nazwę pliku. Przekierowałem więc wejście za pomocą operatora “<” co w rezultacie pozwoliło żeby komenda zwracała tylko liczbę lini. Następnie zadeklarowałem zmienne sterujące grą, tj. status wygranej, czy gra jest uruchomiona, liczbę błednych odpowiedzi, liczbę odgadniętych liter, zamaskowane hasło. Po tych czynnościach zadecydowałem, że napiszę funkcję która będzie sprawdzać czy litery podane przez użytkownika znajdują się w haśle. Tak powstała pierwotna wersja funkcji find\_and\_replace. Przeszedłem do pisania głównej pętli, w funkcj run. Od razu zorientowałem się, że na samym początku powinien być wyświetlany wisielec razem z zamaskowanym hasłem. Umieściłem więc serię funkcji wrongX, która nie robi nic innego jak wyświetla wisielca w danym stanie. Pod spodem widnieje zamaskowane hasło z ewentulnymi odgadniętymi literami i ilość pozostałych prób gracza. Następnie skrypt sprawdza czy ilość wykorzystanych prób nie przekracza maksimum (tutaj wynosi 7 i jest to uzależnione od ilości fukcnji wrongX im więcej kroków do narysowania wisielca tym więcej prób). W końcu następuje zablokowanie działa programu aż do wczytania litery i wywołania funkcji find\_and\_replace. Tutaj zorientowałem się, że mam narazie tylko sprawdzenie czy gracz wykorzystał wszystkie szanse i przegrał. Musiałem więc rozbudować funkcję find\_and\_replace, żeby sprawdzała czy ilość odgadniętych liter jest równa ilośći liter w haśle i jeżeli tak by było to ustawi wynik gracza na wygraną i zakończy grę. Pod funkcją run umieściłem jeszcze wyświetlenie stosownych komunikatów na zakończenie gry, a implementację funkcji dodałem w odpowienim miejscu skryptu. Po przeczytaniu skryptu stwierdziłem, że nie trzymam się jednej konwencji pisania pętli i warunków if. Sprowadziłem wszystkie instrukcje do postaci ze znakiem nowej lini zamiast średnika. Kod wydawał się bardziej spójny. Na tym zakończyłem mój projekt.

Dane Techniczne

Zmienne:

* **dicpl=./slowa.txt** - przechowanie ścieżki do pliku
* **lines=`wc –l < $dicpl`** - przechowanie ilośći lini, opcja–l zwraca ilosc lini pliku. Operator “<” pozwala na pominięcie nazwy pliku w wyniku polecenia wc.
* **word\_number=`shuf –i 1-$lines –n1`** - losowanie liczby. Wykorzystano polecenie shuf z opcją–i (przedział) i ilość losowanych liczb za pomocą -n.
* **word=`sed “${word\_number}q;d” $dicpl`** - słowo-hasło. Polecenie sed zwraca linię pliku pod numerem $word\_number z pliku $dicpl.

Funkcje:

* **find\_and\_replace() -** funkcja pobiera literę jako argument, następnie ustawia zmienną **founded** na false. Zmienna ta przechowuje stan, który opisuje fakt czy podana litera znajduje się w haśle. Następnie uruchomiona zostaje pętla o długośi od 0 do długości słowa. Tworzona jest zmienna **char** i ustawiona jest na i-tą literę w haśle. Jeżeli podana litera jest taka sama jak litera z hasła, to zmienna **founded** ustawiana jest na true, azmienna **game\_word** (tablica) zamienia znak podłogi na daną literę. Dzięki temu użytkownik zobaczy potem odkryte litery. Zwiększony zostaje także licznik odgadniętych liter. Następnie sprawdzony jest warunek ile liter zostało odgadniętych. Jeżeli ilość odgadniętych liter jest równa literom w haśle to znaczy, że całe hasło zostało odkryte i użytkownik wygrał (zmienna **game\_is\_running** zostaje ustawiona na false w celu zakończenia gry). Ostatni warunek wykonywany jest wtedy, gdy użytkownik nie odgadł litery. Wówczas **founded** jest nadal false, a więc zwiekszamy tym zamym licznik błędnych odpowiedzi **wrong\_shoots**.
* **banner() –** prosta funkcja wyświetlająca logo.
* **wrongX() –** szereg funkcji wyświetlających kolejne stany wisielca, gdzie X to liczba od 0 do 7. Zero to początek gdy pusta przestrzeń, a 7 to w pęłni narysowany wisielec oznaczający porażkę.
* **run() –** funkcja uruchamiająca grę. Na początku wykonana jest pętla while (warunek to stan **game\_is\_running)**, która czyści ekran i wyświetla baner. Pod banerem za pomocą instrukcji case i **wrong\_shoots** wyświetlany jest wisielec. Pod nim wyświetlane jest hasło (w postaci znaków podłogi i ewentualnych trafionych liter), a linijkę niżej ilość pozostałych prób. Sprawdzany jest też warunek czy gracz nie przeroczył ilości maksymalnych prób, jeżeli tak jest to zmienna **game\_is\_running** ustawiana jest na false pętla kończy się a gracz przegrywa. W kolenycj krokach pobierana jest litera i wywoływana funkcja **find\_and\_replace $user\_letter.**
* **game\_win(), game\_over()** - dwie funkcję czyszczące ekran i wyświetlające komunikaty na zakończenie gry.