Keybored - sprawozdanie z projektu

Kacper Achramowicz

16 czerwca 2021

Spis treści

1 Wstęp

Zadaniem projektowym postawionym przede mną przez prowadzącego było napisanie gry mobilnej na system Android. Stworzona przeze mnie aplikacja to "Keybored", którą napisałem w Pythonie. Przy tworzeniu GUI korzystałem również z framerwoka kivy. Pliki *.apk tworzyłem korzystając z programu buildozer. Kod źródłowy umieściłem w repozytorium: `https://github.com/Achreko/INF-STS-Ind`

2 Analiza funkcjonalna

2.1 Słownik

przegrana - nienaciśniecię odpowiedniego przycisku w interwale,
wygrana - naciskanie odpowiednich przycisków do czasu, gdy interwał będzie mniejszy niż 0,5 sekundy,
plansza - ekran podzielony na 9 prostokątów równej wielkości, gdzie użytkownik rozgrywa partię,
partia - rozgrywka zaczynająca się od naciśnięcia przycisku start na pierwszym ekranie, lub po wybraniu opcji restart w wypadku wygranej lub przegranej polegająca na naciskaniu podświetlających się na biało przycisków w zmniejszających się interwałach
gameloop - dosłownie pętla gry, czyli powtarzające się akcje stanowiące grę

2.2 Opis funkcjonalności

Program spełnia początkowe założenia z wyjątkiem jednego, a mianowicie przycisku pozwalającego na wyłączenie aplikacji. Funkcjonalność ta okazała się niepotrzebna ze względu na specyfikę wyłączania aplikacji na urządzeniach z systemem Android. Do funkcjonalności należą:
- wyświetlanie ekranu startowego,
- przejście z ekranu startowego do planszy po naciśnięciu przycisku "Start",
- rozegranie partii,
- w wypadku przegranej wyświetlenie odpowiedniego pop-upu z przyciskiem "Restart",
- poinformowanie użytkownika o wygranej odpowiednim pop-upem, który również zawiera przycisk "Restart".

3 Implementacja

Aplikacja zawiera się w plikach manager.kv man.py oraz main.py. Główną trudnością implementacji było odpowiednie połączenie klas napisanych w Pythonie z regułami zapisanymi w kivy language.

3.1 main.py

Ustawia wymagania dotyczące posiadanej wersji kivy oraz łączy wszystkie pliki. Zawiera klasę Keybored(App) odpowiadającą za stworzenie i uruchomienie aplikacji.

3.1.1 Metoda build(self)

Zwraca główny widget(root_widget), a następnie uruchamia aplikację.

3.2 man.py

Tworzy główny widget ładując do niego dane zawarte w pliku manager.kv oraz definiuje metody jego widgetów. Zawiera puste klasy, do których reguły zostały utworzone w kivy language, czyli klasy:
-Man(ScrenManager), która jest głównym widgetem, -MenuScreen(Screen) czyli ekran startowy, -Lost(Popup), czyli okienko przegranej, -Won(Popup), czyli okienko wygranej. Jedyna klasa posiadająca pola i metody w tym pliku to klasa **GameScreen(Screen)**. W ostatniej linii tego pliku wczytywany jest root_widget z pliku manager.kv.

3.2.1 Pole dt

Długość interwału w sekundach.

3.2.2 Pole ct

Licznik naciśnięć poprawnego przycisku. Jeżeli nie jest równy polu *it* to gracz przegrywa.

3.2.3 Pole it

Licznik wyświetleń przycisku. Jeżeli nie jest równy polu *ct* to gracz przegrywa.

3.2.4 Pole flag

Flaga pozwalająca stwierdzić, czy można zacząć gameloop.

3.2.5 Pole prop

Numer przycisku, który należy nacisnąć.

3.2.6 Pole timer

Zegar realizujący interwał.

3.2.7 Metoda restart(self)

Przywraca wartości początkowe polom, przywraca planszę do stanu wejściowego.

3.2.8 Metoda change_flag(self)

Zmienia wartość flag na True oraz wywołuje metodę game_loop(self).

3.2.9 Metoda game_loop(self)

Inicjuje pole timer.

3.2.10 Metoda randomizer(self, interval)

Korzystając z generatora liczb pseduolosowych podświetla odpowiedni przycisk.

3.2.11 Metoda check(self, instance)

Metoda wywoływana po naciśnieciu przycisku, sprawdza, czy został naciśnięty odpowiedni przycisk wywołując przy tym metodę get_id(self, instance) i jeżeli ten warunek nie zostanie spełniony to wywołuje metodę loss(self). Sprawdza też, czy nie nastąpiła wygrana. Jeżeli tak to zatrzymuje timer i wywołuje pop-up Won(Popup).

3.2.12 Metoda get_id(self, instance)

Ustala, który przycisk został naciśnięty.

3.2.13 Metoda loss(self)

Zatrzymuje timer i wywołuje pop-up Loss(Popup).

3.3 manager.kv

W pliku wyznaczam root_widget oraz podpinam do niego pozostałe widgety.
Definiuję również etykietę <GameButton@ToggleButton>, czyli przycisku służące do gry bazujące na przycisku typu ToggleButton. W etykiecie nadaję przyciskom kolor oraz to, że po naciśnięciu wywoływana jest metoda man.gamescreen.check(self, instance).
Nadaję reguły opisujące pop-upy <Lost> oraz <Won> odpowiadające kolejno klasom Loss(Popup) i Won(Popup), które robią w praktyce to samo różniąc się jedynie podpisem oraz podpinam do nich przycisk "Restart".
Tworzę reguły opisujące <MenuScreen> odpowiadający klasie man.menuscreen(Screen).
Do <GameScreen> odpowiadającemu klasie man.gamescreen(Screen) dołączam 9 instancji etykiety <GameButton>.

4 Analiza i wnioski

Analizując wykonaną pracę stwierdzam, że mogłoby być dobrą praktyką przeniesienie pustych klas do oddzielnych plików, ale są też argumenty przeczące temu stwierdzeniu. Musiałbym wtedy w każdym z plików importować from kivy.lang import Builder, aby wczytywać reguły zapisane w kivy language z ciągu znaków, a nie z pliku. Mogłoby to zajmować więcej czasu i pamięci.
Kolejnym wnioskiem jest to, że w celu udoskonalenia GUI można skorzystać jeszcze z kivy material design (kivymd), gdzie znajduje się wiele preconfigurowanych widgetów. W tym projekcie postanowiłem skorzystać jednak z tych utworzonych ręcznie w celu lepszego zrozumienia ich działania.
Wniosek ostatni, a dla mnie jako studenta najważniejszy to jak ważna jest umiejętność czytania dokumentacji. Zaczynając projekt pierwsze reguły w kivy language tworzyłem na przestrzeni dni, gdzie po dokładniejszym zapoznaniu się z dokumentacją problemy, które na początku uznałem za niewarte rozwiązywania ze względu na potrzebny do nich nakład pracy i ich stosunkowy brak wagi, rozwiązałem w ostatnim dniu pisania projektu.