Kamila Kubala
Projekt - Kurs Rozszerzony Języka Python

3CupGame - EV3

8 stycznia 2020

SPIS TREŚCI

O projekcie

W ramach projektu zaprogramowałam robota Lego do popularnej gry w 3 kubki. Gra polega na umieszczeniu kulki pod środkowym kubkiemi wyboru jednego z 4 dostępnych poziomów rozgrywki. Każdy kolejny poziom miesza kubkami coraz szybciej i więcej razy. Po wyborze poziomu, należy poczekać aż zakończy się mieszanie kubków i spróbować odgadnąć pod którym kubkiem znajduje się kulka. RObot będzie w stanie określić czy nasza decyzja była słuszna i pokazać nam zawartość kubka. Miłej zabawy!

Pliki

1. gra.py

   Główny plik gry.

   Na początku program informuje nas o rozpoczęciu pracy wypowiedzianą komendą “Start”.

   ```
   sound.speak("Start")
   ```

   Zmienna boolowska gra, początkowo ustawiona na True. Determinuje działanie pętli while.

   ```
   gra = True
   ```

   W pętli:

   Uruchamiamy funkcję wybierz znajdującą się w pliku *wybor.py*. Pozwala ona wybrać użytkownikowi jeden z czterech dostępnych poziomów gry a wybór zapisać do zmiennej poziom.

   ```
   poziom = wybierz(4)
   ```

   Następnie uruchamiamy funkcję zamieszaj (dostępnej w pliku silniki) zależną od wcześniej ustalonej zmiennej poziom. Funkcja losowo obraca silnikami w ilości oraz z prędkością zależną od poziomu. Zwraca liczbę [1, 2, 3] która określa aktualną pozycję kulki.

   ```
   kulka = zamieszaj(poziom)
   ```

   UWAGA! Przed rozpoczęciem działania programu kulka musi znajdować się pod drugim kubkiem.

   Zagadka zmienna boolowska, przechowuje informacje czy użytkownikowi udało się odgadnąć pozycje kulki. Na początku oczywiście jest fałszywa.

   ```
   zagadka = False
   ```

W pętli wewnętrznej:

Uruchamiamy funkcję tak_nie (z pliku *wybor.py*) w trybie “Guessing?”. Funkcja zwraca bool’a. Jeśli użytkownik nie chce zgadywać dalej betla wewnętrzna zostaje zakończona.

```
if (not tak_nie("Guessing?")):
    break
```

Jęli użytkownik zdecyduje się zgadywać pozwalamy mu wybrać kubek, pod którym jego zdaniem znajduje się kulka. Korzystamy z funkcji wybierz (z pliku *wybor.py*), która zwraca jedną z liczb [1, 2, 3]. Wynik zapisujemy do zmiennej wybor.

```
wybor = wybierz(3)
```

Następnie zostaje wywołana funkcja sprawdź (z pliku *silniki.py*), która jako argumenty przyjmuje wybór użytkownika oraz pozycję kulki. Zwraca listę złożoną z bool’a mówiącego czy użytkownikowi udało się odgadnąć pozycję kulki, oraz aktualną pozycję kulki. (Zauważ, że pozycja kulki może zmienić się podczas sprawdzania).

```
[zagadka, kulka] = sprawdz(wybor, kulka)
```

Jeśli użytkownikowi nie udało się odgadnąć pozycji kulki wracamy do pętli wewnętrznej. Jeśli mu się udało kończymy pętle.

Ostatnim krokiem jest pytanie użytkownika czy chce zagrać ponownie. Korzystamy w tym celu ponownie z funkcji tak_nie (*wybor.py*) w trybie “Play again?”. Wynik działania funkcji wpisujemy do zmiennej gra. Łatwo zauważyć, że jeśli użytkownik wybierze “No” kończymy działanie programu. Wybór “Yes” wraca do początku pętli zewnętrznej.

```
gra = tak_nie("Play again?")
```

2. narzedzia.py

Plik zawiera definicję silników, czujników, ekranu oraz dźwięku z których może korzystać EV3.

- sound - moduł obsługujący dźwięki kostki. Wykorzystuje bibliotekę

```
from ev3dev2.sound import *
```

```
sound = Sound()
```

- ts - definicja czujniku dotyku podpiętego do portu 1.

```
ts = TouchSensor(INPUT_1)
```

- podczerwien - definicja czujnika podczerwieni podpiętego do portu 4. Dzięki niemu jesteśmy w stanie korzystać z pilota.

```
podczerwien = InfraredSensor(INPUT_4)
```

- silnikB, silnikC - odpowiadają za pracę silników wpiętych kolejno do portu A i B. Obracają one mieszadłami do kubków.

```
from ev3dev2.motor import *
```

```
silnikB = LargeMotor(OUTPUT_B)
```

```
silnikC = LargeMotor(OUTPUT_C)
```

- manipulator - silnik średni odpowiadający za podnoszenie i opuszczanie kubków.

```
manipulator = MediumMotor(OUTPUT_A)
```

- ekran - moduł odpowiada za wyświetlanie treści na ekranie.

```
ekran = Display()
```

3. ekran.py

Zawiera funkcje umożliwiającą wyświetlanie treści na ekranie.

Importuje narzedzia sound oraz ekran z pliku *narzedzia.py*.

Korzysta również z biblioteki fontów EV3:

```
import ev3dev2.fonts as fonts
```

a. **`wyswietlanie(tryb, numer):`**

Funkcja przyjmuje dwa argumenty:

- tryb - string, świadczący o aktualnym stanie gry. Np podczas wybierania kubków tryb będzie zawierał “Cup”, a podczas wyboru poziomu “Level”.
- wybor- aktualnie wybierany numer. Np levelu lub kubka lub opcja “Yes/No”.

Na początku funkcja czyli aktualną treść na ekranie.

```
ekran.clear()
```

Następnie u góry ekranu wypisuje w jakim znajdujemy się trybie.

```
ekran.draw.text((10,10), tryb, font=fonts.load('luBS24'))
```

Dalej na środku ekranu wyświetla aktualnie wybieraną opcję.

```
ekran.draw.text((80,60), str(wybor),font=fonts.load('luBS24'))
```

Aktualizuje ekran.

```
ekran.update()
```

Wypowiada jaki wybór został wybrany.

```
sound.speak(str(wybor))
```

4. wybor.py

Zawiera dwie funkcje umożliwiające dokonywanie wyborów przez użytkownika.

Importuje narzędzia sound, ts, podczerwien z pliku *nardzenia.py*

Korzysta również z pliku ekran.py

a. **wybierz(ilosc):**

Funkcja przyjmuje ilosc dostępnych wyborów. W przypadku wyboru poziomów jest to 4, natomiast w przypadku kubków 3.

Z początku wybor równy jest 1.

```
wybor = 1
```

Tryb jest determinowany na podstawie argumentu ilosc.

```
tryb = "LEVEL" if ilosc == 4 else "CUP"
```

Funkcja wyświetla tryb i ilosc na ekranie za pomoca funkcji wyswielanie z pliku ekran.py

```
wyswietlanie(tryb, wybor)
```

Następnie funkcja oczekuje na wciśnięcie przycisku ts , którego definicja znajduje sie w pliku narzedzia.py. Dopóki przycisk nie zostanie wciśnięty mamy możliwość zmiany poziomów za pomocą pilota. Wciśnięcie lewego górnego przycisku na pilocie powoduje zwiększenie poziomu, natomiast wciśnięcie dolnego lewego przycisku powoduje zmniejszenie go.

```
    while (not ts.is_pressed):
        if(podczerwien.top_left(channel=1) and wybor < ilosc):
            wybor += 1
            wyswietlanie(tryb, wybor)
        if(podczerwien.bottom_left(channel=1) and wybor > 1):
            wybor -= 1
            wyswietlanie(tryb, wybor)
    return wybor
```

Zmiany są na bieżąco aktualizowane na ekranie.

Funkcja zwraca numer zwracanego przez użytkownika levelu lub kubka.

b. `tak_nie(tryb):`

Funkcja jest bardzo podobna, jednak kilka istotnych różnic skłoniło mnie do napisania dwóch osobnych funkcji zamiast jednej rozbudowanej.

Istotne różnice to:

- Zamiast wybierania numerów (poziomów) wybieramy spośród dwóch opcji “Tak/Nie”.

```
opcje = ["Yes", "No"]
```

- Naciśnięcie obojętnego (spośród dwóch wcześniej opisanych) przycisku pozwala na zmianę opcji.

```
while (not ts.is_pressed):

        if(podczerwien.top_left(channel=1) or
podczerwien.bottom_left(channel=1)):

            wybor = (wybor + 1) % 2

            wyswietlanie(tryb, opcje[wybor])
```

- Zwracany jest bool

  True - w przypadku chęci kontynuowania

  Falne - w przeciwnym wypadku

5. silniki.py

   Plik zawiera funkcje które obsługują głównie obroty silników.

   Importują narzędzia silnikB, silnikC, manipulator, sound z pliku *narzedzia.py*

   Korzystają z bibliotek random oraz time.

   a. **zamieszaj(poziom):**

      Jako argument przyjmuje poziom gry. Będzie on istotny podczas ustalania ilości i szybkości obrotów silników.

      Na początku ustawiamy pozycję kulki na 2.

```
kulka = 2
```

      Następnie losujemy jedną z liczb [1, 2, 3, 4], każda z liczb będzie obracać jednym z dwóch silników w lewo lub w prawo (razem cztery różne strony).

```
for i in range(random.randint(10*poziom, 10*(poziom+1))):
    strona = random.randint(1, 4)
    if strona == 1:
        kulka = przekrec(silnikB, "B", poziom, 230, -50, kulka)
    elif strona == 2:
        kulka = przekrec(silnikB, "B", poziom, -230, 50, kulka)
    elif strona == 3:
        kulka = przekrec(silnikC, "C", poziom, 230, -50, kulka)
    elif strona == 4:
        kulka = przekrec(silnikC, "C", poziom, -230, 50, kulka)
```

      Funkcja zwraca aktualną pozycję kuli w postaci liczby [1, 2, 3].

b. przekrec(silnik, strona, poziom, stopien, powrot, kulka):

Funkcja obraca silnikiem w określoną stronę, z określoną mocą.

Przyjmuje argumenty:

- silnik - określa który silnik powinien zostać obrócony (A lub B).
- strona - to nazwa obracanego silnika, będzie wykorzystywana do monitorowania kulki.
- poziom - poziom trudności gry, zależy od niego prędkośc obracanego silnika.
- stopień - informacja o ile stopni powinien się obrócić silnik 230 lub -230.
- powrót - definiuje o ile stopni musi wrócić silnik, aby finalnie jego pozycja zmieniła się dokładnie o 180 stopni.
- kulka - aktualna pozycja kulki.

Funkcja dwukrotnie obraca silnikiem. Raz o 230 stopni, aby wypozycjonować kubek idealnie po środku.

```
silnik.on_for_degrees(speed=(poziom*10), degrees=stopien)
```

Drugi raz wraca o 50 stopni, aby pokrętło przyjęło domyślną pozycję 0.0

```
silnik.on_for_degrees(speed=(poziom*10), degrees=powrot)
```

Następnie uruchomiona zostaje funkcja monit_kulki z pliku *kulka.py*. Która aktualizuje pozycję kulki.

```
kulka = monit_kulki(strona, kulka)
```

Funkcja zwraca aktualne położenie kulki.

```
return kulka
```

c. sprawdz(wybor, kulka):

Funkcja pozwala sprawdzić, czy wybrany przez użytkownika kubek zawiera kulkę.

Funkcja przyjmuje dwa argumenty:

- wybor - cyfre [1, 2, 3] oznaczającą wybrany przez użytkownika kubek
- kulka - aktualna pozycja kulki.

Jeśli użytkownik wybierze jeden ze skrajnych kubków (1 lub 3). Zostaje wywołana funkcja przekręć na odpowiednim silniku. Tak aby finalnie wybrany kubek znajdował się na wybranej pozycji. Aktualizujemy przy tym faktyczną pozycję kulki.

```
if wybor == 1:
    kulka = przekrec(silnikB, "B", 2, 230, -50, kulka)
elif wybor == 3:
    kulka = przekrec(silnikC, "C", 2, 230, -50, kulka)
```

Następnie za pomocą manipulatora podnosimy środkowy kubek.

```
manipulator.on_for_degrees(speed=10, degrees=220)
```

Jeśli znajdowała się pod nim kulka kostka pogratuluje użytkownikowi, zamknie kubek i zwróci listę (True, kulka). True oznacza, że użytkownik odgadł pozycje.

```
    if kulka == 2:
        sound.speak("Congratulation!")
        manipulator.on_for_degrees(speed=10, degrees=-220)
        return (True, kulka)
```

Wpp, kosta wyda stosowny komunikat, zamknie kubek, a następnie zwróci (False, kulka). Fałsz oznacza, że użytkownikowi nie udało się odgadnąć pozycji.

```
else:
    sound.speak("Sorry, you are wrong!")
    manipulator.on_for_degrees(speed=10, degrees=-220)
    return (False, kulka)
```

6. kulka.py

Plik zawiera funkcje monit_kulki, który odpowiada za monitorowanie aktualnego położenia kulki.

a. monit_kulki(silnik, kulka):

Funkcja przyjmuje dwa argumenty:

- silnik - informacja którym silnikiem obróciliśmy [silnikA, silnikB].
- kulka - wcześniejsze położenie kulki [1, 2, 3].

Zależnie od tego w którym kubku znajduje się kulka i którym silnikiem obracamy, zmieniamy położenie kulki. Np. Jęli kulka znajduje się w pierwszym kubku, a obracamy silnikiem C jej pozycja się nie zmienia. Jednak jeśli obrócimy silnikiem B jej pozycja zmieni się z 1 na 2.

```
    if (silnik == "B" and kulka == 2) or (silnik == "C" and
kulka == 1):

        return 1

    elif (silnik == "B" and kulka == 1) or (silnik == "C" and
kulka == 3):

        return 2

    elif (silnik == "B" and kulka == 3) or (silnik == "C" and
kulka == 2):

        return 3
```

Pliki testowe

1. Spis plików
    - testy.py

      Główny plik testowy. Jego uruchomienie odpala wszystkie istniejące testy w pozostałych plikach.

    - testy_narzedzia.py

      Plik przechowuje nową implementację narzędzi z których korzystamy w projekcie. To z nich będziemy korzystać podczas testowania poszczególnych funkcji.

    - testy_wybor.py

      Plik zawiera unittesty funkcji z pliku wybor.py. W tym celu korzysta z narzędzi testowych.

    - testy_silniki.py

      Plik zawiera unittesty funkcji z pliku silniki.py. W tym celu korzysta z narzędzi testowych, a także mocków funkcji random.

    - testy_kulka.py

      Plik zawiera unittesty funkcji monit_kulki z pliku kulka.py.

2. Uruchamianie

   Aby włączyć testy zbiorowe należy uruchomić plik testy.py

   ```
   python3 testy.py
   ```

   Można również uruchamiać pojedyncze pliki testowe.

Instrukcja użytkownika

1. Uruchomienie gry

   Aby uruchomić grę należy wybrać plik gra.py znajdujący się w folderze Projekt. Kiedy gra zostanie uruchomiona poprawnie usłyszysz dźwięk “Start”

2. Wybór poziomu

   Następnie zostaniesz poproszony o wybór poziomu. Masz do wyboru cztery różne poziomy. Na każdym kolejnym poziomie silniki obracają się więcej razy i szybciej.

   Aby dokonać zmiany poziomu użyj pilota. Lewy górny przycisk służy do zwiększania poziomu, lewy dolny przycisk służy do jego zmniejszania. NIe możesz zmniejszyć poziomu poniżej 1 ani zwiększyć go powyżej 4. Wybrany poziom wyświetla się na ekranie kostki. Jeśli zdecydowałeś się na swój wybór zatwierdź go za pomocą czujnika dotyku zamontowanego w robocie.

3. Mieszanie

   Jeśli prawidłowo wybrałeś poziom rozpocznie się mieszanie. Zależnie od wybranego poziomu silniki mogą obrócić się od 10 do 50 razy. POczekaj aż ten proces się zakończy i uważnie obserwuj środkowy kubek.

4. Decyzja o próbie zgadywania

   Kiedy mieszanie się zakończy zostaniesz zapytany czy chcesz podjąć próbę zgadywania. Jeśli Wybierzesz opcję “Yes” program przejdzie do następnego punktu. Jeśli jednak nie zdecydujesz się zgadywać, wybierz opcje “No” program przejdzie wtedy od razu do punktu 7 instrukcji. Wybór opcji odbywa się na podobnych zasadach jak przy wyborze opcji. Swój wybór zatwierdzamy czujnikiem dotyku.

5. Wybór kubka

   Jeśli zdecydowałeś się zgadywać program poprosi cię o wybranie jednego z trzech kubków. Wybór odbywa się na tej samej zasadzie co wybieranie poziomu, z tą różnicą że masz tylko 3 dostępne opcje. Zatwierdzamy wybór czujnikiem dotyku

6. Sprawdzenie poprawności wyboru.

   Po wybraniu przez ciebie kubka robot przeniesie go na środkową pozycję i odkryje jego zawartość. Jeśli udało ci się odgadnąć pozycję kulki robot pogratuluje ci i przejdzie do następnego punktu. Jeśli ci się nie udało Robot poinformuje cię że się pomyliłeś, a następnie znów da ci możliwość kontynuowania zgadywania (wracamy do punktu 4)

7. Decyzja o ponownej grze

   Jeśli uda ci się odgadnąć pozycję kulki lub zdecydujesz, że nie chcesz jej dalej zgadywać zostaniesz zapytany o chęć kontynuowania gry.

   Znów wybór odbywa się na tej samej zasadzie co w punkcie 4. Jeśli zdecydujesz się na wybór “Yes” gra rozpocznie się od nowa.

   Jeśli wybierzesz “No” program zakończy się.