Dokumentacja opisująca strukturę programu

Skład zespołu: Tobiasz Mańkowski, Maksymilian Burdziej, Szymon Azarewicz

Moduły

- **from ursina import ***: Importuje moduł Ursina, który jest wykorzystywany w programie.
- **import time**: Importuje moduł time, który umożliwia manipulację czasem.
- **import random**: Importuje moduł random, który umożliwia generowanie liczb losowych.
- **import datetime**: Importuje moduł datetime, który umożliwia manipulację datą i czasem.

Klasa Timer

- **def __init__(self)**: Konstruktor inicjalizuje obiekt klasy Timer. Ustawia początkowe wartości atrybutów **wlacznik**, **poczatek** i **koncowy_czas**.
- **def start(self)**: Metoda rozpoczyna pomiar czasu. Ustawia atrybut **wlacznik** na True i zapisuje bieżący czas w atrybucie **poczatek**.
- **def stop(self)**: Metoda zatrzymuje pomiar czasu. Ustawia atrybut **wlacznik** na False.
- **def podaj_czas(self)**: Metoda zwraca upływający czas od rozpoczęcia pomiaru. Jeśli **wlacznik** jest True, oblicza różnicę między bieżącym czasem a **poczatek** i zwraca ją. W przeciwnym razie zwraca wartość atrybutu **koncowy_czas**.
- **def pauzowanie(self)**: Metoda służy do pauzowania pomiaru czasu. Jeśli **wlacznik** jest True, wywołuje metodę **stop()**, w przeciwnym razie wywołuje metodę **start()**.

```
class Timer:
    def __init__(self):
        self.wlacznik=False
        self.poczatek=0
        self.koncowy_czas=0

    def start(self):
        self.wlacznik=True
        self.poczatek= time.time()

    def stop(self):
        self.wlacznik=False


    def podaj_czas(self):
        if self.wlacznik==True:
            self.koncowy_czas= time.time()- self.poczatek
            return time.time()- self.poczatek
        else:
            return self.koncowy_czas
    def pauzowanie(self):
        if self.wlacznik==True:
                self.stop()
        else:
                self.start()
```

Klasa Gra

Klasa Gra jest klasą pochodną z biblioteki Ursina i reprezentuje główną logikę gry.

Oto opis poszczególnych elementów klasy:

- **__init__(self)**: Inicjalizuje klasę, tworzy instancję **Timer** do mierzenia czasu gry, inicjalizuje listę **wykonane_ruchy** i tworzy przyciski oraz teksty dla interfejsu użytkownika.

```
def __init__(self):
    super().__init__()
    # tryb
    # 1 - ruch
    # 2 - ułóż
    # 3 - cofnij
    # 4 - mieszaj
    self.RUCH = 1
    self.ULOZ = 2
    self.COFNIJ = 3
    self.MIESZAJ =4
    self.rotation_direction = 1
    window.fullscreen = False
    Entity(model='sphere', scale=10000, texture='Zenek1', double_sided=True)
    EditorCamera()

    camera.world_position = (0, 0, -15)
    self.load_game()

    self.wykonane_ruchy=[]
    self.stoper= Timer()

    self.Licznik_Czasu=Button(text="Mierz czas", color=color.red, scale=.13, position = (0.45, -0.4), on_click= self.stoper.pauzowanie)
    self.Cofanie=Button(text="Cofnij ruch", color=color.red, scale=.13, position = (0.60, -0.4), on_click= self.cofnij_ruch)
    self.Ukladanie=Button(text="Ułóż kostkę!", color=color.red, scale=.13, position = (0.75, -0.4), on_click= self.uloz_kostke)
    self.Losowanie=Button(text="Pomieszaj!", color=color.red, scale=.13, position = (0.45, -0.25), on_click= self.pomieszaj_kostke)
    self.czas= Text(text=f"Twój czas: 0.0 sekund", color= color.red, position=(-0.50, 0.4), size=.05)
    self.Zapisz=Button(text="Zapisz", color=color.red, scale=.13, position=(0.6, -0.25), on_click=self.zapisz_ruchy_do_pliku)
    self.Zapisz=Button(text="Wczytaj", color=color.red, scale=.13, position=(0.75, -0.25), on_click=self.wczytaj_ruchy_z_pliku)
    self.update()
```

- **zapisz_ruchy_do_pliku(self)**: Zapisuje wykonane ruchy do pliku tekstowego "zapis.txt".
- **wczytaj_ruchy_z_pliku(self)**: Wczytuje ruchy z pliku tekstowego "zapis.txt" i odtwarza je na planszy.
- **update(self)**: Aktualizuje stan gry, w tym czas gry, i wywołuje samą siebie rekurencyjnie.
- **pomieszaj_kostke(self)**: Miesza kostkę poprzez losowe wybieranie stron i kierunków obrotu.
- **cofnij_ruch(self)**: Cofa ostatni wykonany ruch przez odwrócenie kierunku obrotu.
- **uloz_kostke_zostaw_liste(self)**: Układa kostkę na planszy na podstawie listy wykonanych ruchów.
- **uloz_kostke(self)**: Układa kostkę na planszy i usuwa listę wykonanych ruchów.

```
def uloz_kostke_zostaw_liste(self):

    if len(self.wykonane_ruchy)!=0:


        for i in range(int(len(self.wykonane_ruchy)/2)):
            stronka=self.wykonane_ruchy[2*i]
            kierunek=int(self.wykonane_ruchy[2*i+1])
            self.obroc_kostke(stronka, kierunek, self.ULOZ)

def uloz_kostke(self):

    if len(self.wykonane_ruchy)!=0:


        while len(self.wykonane_ruchy)>0:
                stronka=self.wykonane_ruchy[-2]
                kierunek=int(self.wykonane_ruchy[-1])
                self.obroc_kostke(stronka, -kierunek, self.ULOZ)
                self.wykonane_ruchy.pop()
                self.wykonane_ruchy.pop()
```

- **load_game(self)**: Ładuje grę, tworzy elementy interfejsu użytkownika i ustawia początkowe wartości.

```
def load_game(self):
    self.stworz_pozycje_kosteczek()
    self.poboczne = [Entity(model='rubik', texture='Cube', position = pos) for pos in self.SIDE_POSITIONS]
    self.main_cube = Entity(model='rubik', collider='box', texture= 'Chess_Board')
    self.osie_obrotu = {'ZIELONA': 'x', 'NIEBIESKA': 'x', 'BIALA': 'y', 'ZOLTA': 'y', 'CZERWONA': 'z', 'POMARANCZOWA': 'z'}
    self.krawedzie_kostki = {'ZIELONA': self.LEFT, 'ZOLTA': self.BOTTOM, 'NIEBIESKA': self.RIGHT, 'CZERWONA': self.FACE, 'POMARANCZOWA': self.BACK, 'BIALA': self.TOP}
    self.animation_time = 0.5
    self.tworzenie_hitboxow()
    self.blokada_ruchu =False
    self.spectator_mode= True


    self.opcje = Text( origin = (0, 16), color= color.red)
    self.zmien_tryb_gry()
```

- **obroc_kostke(self, nazwa_strony, stopnie, tryb)**: Obraca kostkę na podstawie nazwy strony, stopni obrotu i trybu ruchu.

```
def obroc_kostke(self, nazwa_strony, stopnie, tryb):
    if tryb==1 or tryb==4: #zwykly ruch / #ruch mieszania (bez animacji)
        self.blokada_ruchu = True
        cube_positions= self.krawedzie_kostki[nazwa_strony]
        os_obrotu = self.osie_obrotu[nazwa_strony]
        stopnie= stopnie
        self.zmien_nadrzedna()
        for cube in self.poboczne:
            if cube.position in cube_positions:
                cube.parent= self.main_cube
                if tryb == 1:
                    eval(f'self.main_cube.animate_rotation_{os_obrotu}({stopnie}, duration = self.animation_time)')
                elif tryb ==4:
                    exec(f'self.main_cube.rotation_{os_obrotu} = {stopnie}')
        invoke(self.blokada_ruchu_gracza, delay= self.animation_time + 0.1)
        self.wykonane_ruchy.append(nazwa_strony)
        self.wykonane_ruchy.append(stopnie)
    elif tryb==2: #dla ukladania calej
        cube_positions= self.krawedzie_kostki[nazwa_strony]
        os_obrotu = self.osie_obrotu[nazwa_strony]
        stopnie= stopnie
        self.zmien_nadrzedna()
        for cube in self.poboczne:
            if cube.position in cube_positions:
                cube.parent= self.main_cube
                exec(f'self.main_cube.rotation_{os_obrotu} = {stopnie}')
    elif tryb==3: #dla cofania
        cube_positions= self.krawedzie_kostki[nazwa_strony]
        os_obrotu = self.osie_obrotu[nazwa_strony]
        stopnie= stopnie
        self.zmien_nadrzedna()
        for cube in self.poboczne:
            if cube.position in cube_positions:
                cube.parent= self.main_cube
                eval(f'self.main_cube.animate_rotation_{os_obrotu}({stopnie}, duration = 0.1)')
```

- **blokada_ruchu_gracza(self)**: Blokuje możliwość ruchu gracza na określony czas.
- **zmien_tryb_gry(self)**: Zmienia tryb gry między swobodnym a układaniem.
- **zmien_nadrzedna(self)**: Zmienia rodzica kostek, umożliwiając swobodne ich obracanie.
- **stworz_hitboxa(self, nazwa, pozycja, skala)**: Tworzy hitbox do rejestrowania kliknięć na kostce.

```
def stworz_hitboxa(self, nazwa, pozycja, skala):
    self.hitbox = Entity(name=nazwa, position=pozycja, model='rubik', scale= skala, collider= 'box', visible=False)

def tworzenie_hitboxow(self):
    self.LG= self.stworz_hitboxa("czerwony_LG", pozycja=(-1, 1 , -1.5), skala=(1, 1, 0.1))
    self.PG= self.stworz_hitboxa("czerwony_PG", pozycja=(1, 1 , -1.5), skala=(1, 1, 0.1))
    self.LD= self.stworz_hitboxa("czerwony_LD", pozycja=(-1,-1 , -1.5), skala=(1, 1, 0.1))
    self.PD= self.stworz_hitboxa("czerwony_PD", pozycja=(1, -1 , -1.5), skala=(1, 1, 0.1))
    self.LG= self.stworz_hitboxa("zielony_LG", pozycja=(-1.5, 1 , 1), skala=(0.1, 1, 1))
    self.PG= self.stworz_hitboxa("zielony_PG", pozycja=(-1.5, 1 , -1), skala=(0.1, 1, 1))
    self.LD= self.stworz_hitboxa("zielony_LD", pozycja=(-1.5, -1 , 1), skala=(0.1, 1, 1))
    self.PD= self.stworz_hitboxa("zielony_PD", pozycja=(-1.5, -1 , -1), skala=(0.1, 1, 1))
    self.LG= self.stworz_hitboxa("niebieski_LG", pozycja=(1.5, 1 , -1), skala=(0.1, 1, 1))
    self.PG= self.stworz_hitboxa("niebieski_PG", pozycja=(1.5, 1 , 1), skala=(0.1, 1, 1))
    self.LD= self.stworz_hitboxa("niebieski_LD", pozycja=(1.5, -1 , -1), skala=(0.1, 1, 1))
    self.PD= self.stworz_hitboxa("niebieski_PD", pozycja=(1.5, -1 , 1), skala=(0.1, 1, 1))
    self.LG= self.stworz_hitboxa("pomaranczowy_LG", pozycja=(1, 1 , 1.5), skala=(1, 1, 0.1))
    self.PG= self.stworz_hitboxa("pomaranczowy_PG", pozycja=(-1, 1 , 1.5), skala=(1, 1, 0.1))
    self.LD= self.stworz_hitboxa("pomaranczowy_LD", pozycja=(1,-1 , 1.5), skala=(1, 1, 0.1))
    self.PD= self.stworz_hitboxa("pomaranczowy_PD", pozycja=(-1, -1 , 1.5), skala=(1, 1, 0.1))
    self.LG= self.stworz_hitboxa("void_1", pozycja=(0, 0 , 0), skala=(3, 3, 0.1))
    self.LG= self.stworz_hitboxa("void_2", pozycja=(0, 0 , 0), skala=(3, 0.1, 3))
    self.LG= self.stworz_hitboxa("void_3", pozycja=(0, 0 , 0), skala=(0.1, 3, 3))
    self.LG= self.stworz_hitboxa("void_4", pozycja=(0, 1.5 , 0), skala=(3, 0.1, 3))
    self.LG= self.stworz_hitboxa("void_5", pozycja=(0, -1.5 , 0), skala=(3, 0.1, 3))
```

- **input(self, key):** obsługuje interakcję z użytkownikiem, reagując na wciśnięcie klawiszy myszy. Jeśli klawisz należy do zestawu 'mouse1 mouse3' i ustawione flagi blokada_ruchu i spectator_mode są False, następuje sprawdzenie kolizji myszy. Jeśli kolizja występuje na odpowiednich stronach kostki, wywoływana jest metoda obroc_kostke z odpowiednimi argumentami. Jeśli klawisz to 'mouse2', wywoływana jest metoda zmien_tryb_gry.

```
def input(self, key):
    super().input(key)

    if key in 'mouse1 mouse3' and self.blokada_ruchu==False and self.spectator_mode==False:

        stopnie = 90

        for kontakt in mouse.collisions:
                nazwa_strony= kontakt.entity.name
                if ((key== 'mouse1' or key=='mouse3') and nazwa_strony in 'void_1 void_2 void_3 void_4 void_5'):
                    break

                if ((key== 'mouse1' or key=='mouse3') and nazwa_strony in 'czerwony_LG czerwony_PG czerwony_LD czerwony_PD'):

                    if (key=='mouse1' and nazwa_strony =='czerwony_LG'):
                        self.obroc_kostke('ZIELONA', stopnie, self.RUCH)
                    elif (key=='mouse1' and nazwa_strony =='czerwony_LD'):
                        self.obroc_kostke('ZIELONA', -stopnie, self.RUCH)
                    elif (key=='mouse1' and nazwa_strony =='czerwony_PG'):
                        self.obroc_kostke('NIEBIESKA', stopnie, self.RUCH)
                    elif (key=='mouse1' and nazwa_strony =='czerwony_PD'):
                        self.obroc_kostke('NIEBIESKA', -stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='czerwony_PG'):
                        self.obroc_kostke('BIALA', -stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='czerwony_LG'):
                        self.obroc_kostke('BIALA', stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='czerwony_PD'):
                        self.obroc_kostke('ZOLTA', -stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='czerwony_LD'):
                        self.obroc_kostke('ZOLTA', stopnie, self.RUCH)

                    break
                elif ((key== 'mouse1' or key=='mouse3') and nazwa_strony in 'zielony_LG zielony_PG zielony_LD zielony_PD'):
                    if (key=='mouse1' and nazwa_strony =='zielony_LG'):
                        self.obroc_kostke('POMARANCZOWA', stopnie, self.RUCH)
                    elif (key=='mouse1' and nazwa_strony =='zielony_LD'):
                        self.obroc_kostke('POMARANCZOWA', -stopnie, self.RUCH)
                    elif (key=='mouse1' and nazwa_strony =='zielony_PG'):
                        self.obroc_kostke('CZERWONA', stopnie, self.RUCH)
                    elif (key=='mouse1' and nazwa_strony =='zielony_PD'):
                        self.obroc_kostke('CZERWONA', -stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='zielony_PG'):
                        self.obroc_kostke('BIALA', -stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='zielony_LG'):
                        self.obroc_kostke('BIALA', stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='zielony_PD'):
                        self.obroc_kostke('ZOLTA', -stopnie, self.RUCH)
                    elif (key=='mouse3' and nazwa_strony =='zielony_LD'):
                        self.obroc_kostke('ZOLTA', stopnie, self.RUCH)
```

```
        break
    elif ((key== 'mouse1' or key=='mouse3') and nazwa_strony in 'pomaranczowy_LG pomaranczowy_PG pomaranczowy_LD pomaranczowy_PD'):
        if (key=='mouse1' and nazwa_strony =='pomaranczowy_LG'):
            self.obroc_kostke('NIEBIESKA', -stopnie, self.RUCH)
        elif (key=='mouse1' and nazwa_strony =='pomaranczowy_LD'):
            self.obroc_kostke('NIEBIESKA', stopnie, self.RUCH)
        elif (key=='mouse1' and nazwa_strony =='pomaranczowy_PG'):
            self.obroc_kostke('ZIELONA', -stopnie, self.RUCH)
        elif (key=='mouse1' and nazwa_strony =='pomaranczowy_PD'):
            self.obroc_kostke('ZIELONA', stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='pomaranczowy_PG'):
            self.obroc_kostke('BIALA', -stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='pomaranczowy_LG'):
            self.obroc_kostke('BIALA', stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='pomaranczowy_PD'):
            self.obroc_kostke('ZOLTA', -stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='pomaranczowy_LD'):
            self.obroc_kostke('ZOLTA', stopnie, self.RUCH)

        break
    elif ((key== 'mouse1' or key=='mouse3') and nazwa_strony in 'niebieski_LG niebieski_PG niebieski_LD niebieski_PD'):
        if (key=='mouse1' and nazwa_strony =='niebieski_LG'):
            self.obroc_kostke('CZERWONA', -stopnie, self.RUCH)
        elif (key=='mouse1' and nazwa_strony =='niebieski_LD'):
            self.obroc_kostke('CZERWONA', stopnie, self.RUCH)
        elif (key=='mouse1' and nazwa_strony =='niebieski_PG'):
            self.obroc_kostke('POMARANCZOWA', -stopnie, self.RUCH)
        elif (key=='mouse1' and nazwa_strony =='niebieski_PD'):
            self.obroc_kostke('POMARANCZOWA', stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='niebieski_PG'):
            self.obroc_kostke('BIALA', -stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='niebieski_LG'):
            self.obroc_kostke('BIALA', stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='niebieski_PD'):
            self.obroc_kostke('ZOLTA', -stopnie, self.RUCH)
        elif (key=='mouse3' and nazwa_strony =='niebieski_LD'):
            self.obroc_kostke('ZOLTA', stopnie, self.RUCH)
        break
```

- stworz_pozycje_kosteczek: inicjalizuje pozycje kostek na różnych ścianach kostki. Tworzone są zbiory punktów dla lewej, dolnej, przedniej, tylnej, prawej i górnej ściany kostki.

```
def stworz_pozycje_kosteczek(self):
    self.LEFT = {Vec3(-1, y, z) for y in range(-1, 2) for z in range(-1, 2)}
    self.BOTTOM = {Vec3(x, -1, z) for x in range(-1, 2) for z in range(-1, 2)}
    self.FACE = {Vec3(x, y, -1) for x in range(-1, 2) for y in range(-1, 2)}
    self.BACK = {Vec3(x, y, 1) for x in range(-1, 2) for y in range(-1, 2)}
    self.RIGHT = {Vec3(1, y, z) for y in range(-1, 2) for z in range(-1, 2)}
    self.TOP = {Vec3(x, 1, z) for x in range(-1, 2) for z in range(-1, 2)}
    #self.SIDE_POSITIONS = self.BOTTOM |self.BACK
    self.SIDE_POSITIONS = self.LEFT | self.BOTTOM | self.FACE | self.BACK | self.RIGHT | self.TOP
```

Interfejs użytkownika:

Interfejs użytkownika w tym programie składa się z następujących elementów:

1. Okno aplikacji: Aplikacja tworzy okno, w którym wyświetlany jest interfejs użytkownika.
2. Przyciski:
    - Przycisk "Mierz czas": Po kliknięciu rozpoczyna pomiar czasu gry.
    - Przycisk "Cofnij ruch": Po kliknięciu cofa ostatnio wykonany ruch.
    - Przycisk "Ułóż kostkę!": Po kliknięciu układa kostkę w jej początkowym ułożeniu.
    - Przycisk "Pomieszaj!": Po kliknięciu mieszaje kostkę, wykonując losowe ruchy.
    - Przycisk "Zapisz": Po kliknięciu zapisuje wykonane ruchy do pliku tekstowego.
    - Przycisk "Wczytaj": Po kliknięciu wczytuje zapisane ruchy z pliku tekstowego.

3. Tekst "Twój czas": Wyświetla aktualny czas gry.
4. Kamera: Dostarcza podgląd sceny gry.

Interakcje użytkownika:

- Kliknięcie przycisku "Mierz czas" rozpoczyna pomiar czasu gry.
- Kliknięcie przycisku "Cofnij ruch" powoduje cofnięcie ostatnio wykonanego ruchu.
- Kliknięcie przycisku "Ułóż kostkę!" powoduje ułożenie kostki w jej początkowe ułożenie.
- Kliknięcie przycisku "Pomieszaj!" powoduje wymieszanie kostki, wykonując losowe ruchy.
- Kliknięcie przycisku "Zapisz" zapisuje wykonane ruchy do pliku tekstowego.
- Kliknięcie przycisku "Wczytaj" wczytuje zapisane ruchy z pliku tekstowego.

Wizualizacja:

- Kostka Rubika jest wyświetlana na ekranie jako główny element gry.
- Przyciski i tekst są wyświetlane w odpowiednich miejscach na ekranie, umożliwiając interakcję użytkownika.