

Dokumentacja projektu

Prosta gra turowa

Wykonał: Tomasz Marchela 109787

1. Opis projektu

Stworzenie gry turowej na podstawie gry przeglądarkowej Sword and Sandals z nieco uproszczoną formułą, jednakże z dodaniem własnych mechanik a wszystko działające w terminalu. Realizacja projektu odbyła się w pythonie przy użyciu biblioteki Asciiatics oraz PyGame.

Główne założenia: Wymyślenie formuły gry turowej którą można przenieść do terminala windowsowego,linuxowego itd oraz stworzenie widoku graficznego, a następnie zaimplementowanie jej mając na uwadze wszystkie ograniczenia związane z używaniem tylko znaków ascii oraz poznanie działania pygame.

Istota aplikacji: Umożliwienie graczom satysfakcjonujących mechanik gry turowej i zilustrowanie to za pomocą terminala jak i przedstawienie to za pomocą widoku graficznego z użyciem pygame. Dzięki użyciu modelu mvc jedyne co trzeba było zmienić to widok.

2. Opis funkcjonalności

- Poznanie informacji o grze w menu gry. Jej ustawienia i jak grać.
- Trzy różne sklepy w których możemy się zaopatrzyć w potrzebny nam ekwipunek.
 - ❖ Sklep z zbrojami które redukują zadawane obrażenia przez przeciwników.
 - ❖ Sklep z mieczami które zwiększają zadawane obrażenia.
 - ❖ Sklep z ulepszeniami które zwiększają nasze statystyki.
- Trzy walki z przeciwnikami po których pokonaniu wygrywamy grę.
- Po każdej walce otrzymujemy nagrodę w postaci złota możemy przejść do sklepu i rozpocząć następną walkę.
- Po przegranej walce możemy rozpocząć grę od nowa.
- Wszystkie akcje ozdobione ascii art oraz prostymi animacjami.
- Dziesięć różnych akcji podczas walki.
 - ❖ Trzy różne ataki wręcz które im większe mają obrażenia tym mniejszą szansę na trafienie.
 - ❖ Trzy czary które działają różnie na różnych wrogów.

- ❖ Odpoczynek który przywraca nam zdrowie i manę.

3.1. Zagadnienia projektowe w wersji z terminalem

Wykorzystanie biblioteki Asciiatics.

Cała struktura widoków w programie dzieli się na sceny każda czynność w grze jest oddzielną sceną. Każda scena ma przypisane do siebie efekty które dzięki specyfiki gry mogą być predefiniowane. Każda scena składa się z efektów zaimplementowanych w asciimatics. Takie jak gwiazdki i świecący napis w menu gry które możemy dowolnie edytować.



Kod sceny menu:

```
effectsMenu = [  
    Cycle(  
        screen,  
        FigletText("PALLANDIN", font='standard'),  
        screen.height // 2 -15),  
    Cycle(  
        screen,  
        FigletText("VS", font='standard'),  
        screen.height // 2 -10),  
    Cycle(  
        screen,  
        FigletText("MONSTERS", font='standard'),  
        screen.height // 2 -5),  
    Stars(screen,250),  
    MenuObject  
]  
scenes.append(Scene(effectsMenu,-1,name="MENU"))
```

Następnym składnikiem scen są ramki do których jesteśmy dodawać widgety takie jak przyciski, listy do których możemy pisać własne metody i logikę. Przykładowy kod ramki:

```

class FightFrame(Frame):
    # marchlak
    def __init__(self, screen):
        super().__init__(screen,
                          7,
                          screen.width-2,
                          can_scroll=False,
                          title="DUNGEON",
                          x=1, y=screen.height-7
                          )

        self.palette = get_palette()
        layout = Layout([1,1,1,1], fill_frame=True)
        self.add_layout(layout)
        layout.add_widget(Button("ATTACK", self._attack, add_box=False), 0)
        layout.add_widget(Button("MAGIC", self._magic, add_box=False), 1)
        layout.add_widget(Button("REST", self._rest, add_box=False), 2)
        layout.add_widget(Divider(line_char=""), 1)
        layout.add_widget(Button("BACK", self._back, add_box=False), 1)
        PlayerHpValue = self.get_player_hp_text()
        self.player_hp_label = Label(PlayerHpValue)
        layout.add_widget(Label("PLAYER STATS"), 3)
        layout.add_widget(self.player_hp_label, 3)
        MonsterHpValue = self.get_monster_hp_text()
        self.monster_hp_label = Label(MonsterHpValue)
        self.manalabel = Label(self.get_player_mana_text())
        layout.add_widget(self.manalabel, 3)
        layout.add_widget(Label("ENEMY STATS"), 3)
        layout.add_widget(self.monster_hp_label, 3)
        self.fix()

1 usage # marchlak
@staticmethod
def _back():
    raise NextScene("MENU")

```

```

1 usage  📄 marchlak
@staticmethod
def _attack():
    raise NextScene(FightController.scene_controller("ATTACKTYPE"))
1 usage  📄 marchlak
def _rest(self):
    FightController.rest()
    raise NextScene(FightController.scene_controller("REST"))

2 usages  📄 marchlak
def get_player_hp_text(self):
    #print("HP {}/{}".format(FightController.getPlayerHP(), FightController.getMaxPlayerHP()))
    return "HP {}/{}".format(FightController.getPlayerHP(), FightController.getMaxPlayerHP())

2 usages  📄 marchlak
def get_monster_hp_text(self):
    #print("HP {}/{}".format(FightController.getMonsterHP(), FightController.getMaxMonsterHP()))
    return "HP {}/{}".format(FightController.getMonsterHP(), FightController.getMaxMonsterHP())
2 usages  📄 marchlak
def get_player_mana_text(self):
    return "Mana {}/{}".format(FightController.get_mana(), FightController.get_maxmana())

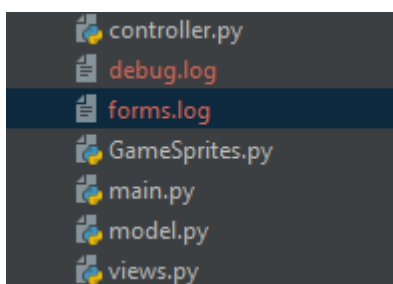
1 usage  📄 marchlak
def update_hp_label(self):
    self.player_hp_label.text = self.get_player_hp_text()
    self.monster_hp_label.text = self.get_monster_hp_text()
    self.mana_label.text = self.get_player_mana_text()

📄 marchlak
def _update(self, frame_no):
    # Refresh the list view if needed
    self.update_hp_label()
    super(FightFrame, self)._update(frame_no)

```

Ostatnim elementem sceny są Sprite pod którymi kryją się asciiart które dzięki klasie path możemy ruszać dodać ich kilka renderów które zawierają ich poszczególne klatki które mogą się zmieniać i funkcje w jakiej kolejności mogą być renderowane.

Logika samej gry jest prosta i przejrzysta. Cały program został podzielony w strukturze mvc co ułatwi przekształcenie go w wersję z interfejsem graficznym.



3.2 Zagadnienia projektowe związane z interfejsem graficznym.

Tworzenie gier w pygame polega na rysowaniu klatek na obrazie w moim przypadku 60 klatek na sekundę dwuwymiarowych obrazów. Oraz interakcję z nimi przez sprawdzanie ich pozycji na ekranie i inputu użytkownika. Oto przykładowy kod jednej ze scen.

1 usage marchlak

```
def fight():  
    BUTTONSTATE = "MENU"  
    death_timer = 0  
    action_timer = 1000  
    attack_timer = 80  
    monster_hp_timer = 1000  
    player_hp_timer = 1000  
    monster_hp = FightController.getMaxMonsterHP()  
    player_hp = FightController.getMaxPlayerHP()  
    if(FightController._iterator == 0):  
        current_monster = skeleton  
        FIGHTBACKGROUND = pygame.image.load("assets/reaperfightbackground.png")  
    if (FightController._iterator == 1):  
        current_monster = bringer  
        FIGHTBACKGROUND = pygame.image.load("assets/Backgroundbringer.png")  
        attack_timer = 120  
    if (FightController._iterator == 2):  
        current_monster = demon  
        FIGHTBACKGROUND = pygame.image.load("assets/hell.png")  
        paladin.change_position(x: 550, y: 740)  
        attack_timer = 120
```

inicjalizacja potrzebnych zmiennych i grafik.

```

while True:
    clock.tick(fps)
    #timers
    death_timer += 1
    action_timer += 1
    player_hp_timer += 1
    monster_hp_timer +=1
    #classdefine
    FIGHT_MOUSE_POS = pygame.mouse.get_pos()
    SCREEN.fill("Black")
    SCREEN.blit(FIGHTBACKGROUND,(0,0))
    PLAY_ATTACK = Button(image=BUTTON, pos=(660, 1000), text_input="ATTACK", font=get_font(25), base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_MAGIC = Button(image=BUTTON, pos=(960, 1000), text_input="MAGIC", font=get_font(25), base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_REST = Button(image=BUTTON, pos=(1260, 1000), text_input="REST", font=get_font(25), base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_ATTACK_STRONG = Button(image=BUTTON, pos=(1100, 1000), text_input="STRONG", font=get_font(25), base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_ATTACK_NORMAL = Button(image=BUTTON, pos=(800, 1000), text_input="NORMAL", font=get_font(25),
        base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_ATTACK_FAST = Button(image=BUTTON, pos=(500, 1000), text_input="FAST", font=get_font(25),
        base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_MAGIC_FIRE = Button(image=BUTTON, pos=(500, 1000), text_input="FIRE", font=get_font(25),
        base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_MAGIC_HOLY = Button(image=BUTTON, pos=(800, 1000), text_input="HOLY", font=get_font(25),
        base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_MAGIC_THUNDER = Button(image=BUTTON, pos=(1100, 1000), text_input="THUNDER", font=get_font(25),
        base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_BACK_BACK = Button(image=BUTTON, pos=(1400, 1000), text_input="BACK", font=get_font(25),
        base_color="#00FFFFFF",
        hovering_color="Green")
    PLAY_BACK_TO_MENU = Button(image=None, pos=(960, 620), text_input="RESTART", font=get_font(25),
        base_color="#00FFFFFF",

```

Główna pętla i inicjalizacja przycisków

```

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    if event.type == pygame.MOUSEBUTTONDOWN:
        if PLAY_ATTACK.checkForInput(FIGHT_MOUSE_POS):
            BUTTONSTATE = "ATTACK"
        if PLAY_MAGIC.checkForInput(FIGHT_MOUSE_POS):
            BUTTONSTATE = "MAGIC"
        if PLAY_BACK_BACK.checkForInput(FIGHT_MOUSE_POS):
            BUTTONSTATE = "MENU"
        if PLAY_ATTACK_FAST.checkForInput(FIGHT_MOUSE_POS):
            BUTTONSTATE = "MENU"
            action_timer = 0
            FightController.setAttacktype("FAST")
            paladin.set_action(1)
            FightController.playerTurn("WEAPON")
            monster_hp_timer = 0
            if not FightController.isMonsterDead:
                player_hp_timer = 0
                current_monster.set_action(2)
                FightController.enemyTurn()
                current_monster.set_action(1)
                if not FightController.isPlayerDead:
                    paladin.set_action(4)
                else:
                    paladin.set_action(3)
                    death_timer = 0
            else:
                current_monster.set_action(3)
                death_timer = 0
        if PLAY_ATTACK_NORMAL.checkForInput(FIGHT_MOUSE_POS):
            BUTTONSTATE = "MENU"
            action_timer = 0
            FightController.setAttacktype("NORMAL")
            paladin.set_action(1)
            FightController.playerTurn("WEAPON")

```

Czytanie inputu użytkownika.

Instrukcja

1. Pobierz pliki Źródłowe gry z CEZ2 lub <https://github.com/Marchlak/PALANDINSANDMONSTERS..>
2. Pobierz python w wersji 3.9
3. Zainstaluj bibliotekę asciimatics "pip install asciimatics" oraz pygame "pip install pygame"
3. Uruchom plik main.py za pomocą python main.py z argumentem ascii lub graphic

Wnioski

Dzięki temu projektowi nauczyłem się wiele o pythonie ale co więcej jestem w stanie tworzyć

multiplatformowe gui dzięki bibliotece asciimatics a także spróbować swoich sił w tworzeniu gry platformowej w pełnym tego słowa znaczeniu. Projekt dał mi także wiele frajdy gdy mogłem odtworzyć swoje gry dzieciństwa takie jak Sword And Sandals lub For The King własnoręcznie.

Samoocena

Projekt oceniam za udany. Udało się mi osiągnąć każdy cel jaki sobie wyznaczyłem jedynie niektórych rozczarowywać mogą animacje walki w widoku ascii nie zadowalać może także widok sklepu jednak spowodowane jest to brakiem dobrych darmowych grafik jednak taka jest konwencja gatunku gier typu battler turowy plus ograniczenia związane w tworzeniu spritów w ascii. Projekt zasługuje na ocenę 5 lub 4.5.

Źródła grafik:

<https://itch.io/>

Dalle ai

<https://www.asciart.eu/>