Gra gatunku RPG akcji stworzona w języku Python z wykorzystaniem biblioteki Pygame

Błażej Szwichtenberg, indeks: 271397

Maj 2024

Spis treści

1 Opis projektu

Jako projekt zdecydowałem się zrobić prostą grę RPG akcji w Pythonie przy użyciu modułu Pygame. Gra inspirowana jest klasycznymi grami przygodowymi z otwartym świecie, w których gracz wciela się w rolę bohatera, który musi odkrywać tajemnice i pokonać wrogów w pełnym przygód świecie.

1.1 Główne mechaniki i elementy gry:

1. **Bohater i sterowanie:**
   Gracz kieruje bohaterem, który może poruszać się w czterech kierunkach. Bohater może atakować i rzucać zaklęcia. Zaimplementowane zostały kolizje i niewidzialne ściany w granicach mapy.

2. **Świat gry:**
   Gra odbywa się w otwartym świecie opartym na danych mapy z zewnętrznych plików csv.

3. **Przeciwnicy i walka:**
   W grze występują 3 różne rodzaje przeciwników, każdy z własnymi statystykami i atakami. Za pokonywanie ich otrzymuje się punkty doświadczenia, za które mozna ulepszać postać.

4. **Grafika i muzyka:**
   Gra posiada autorską grafikę typu pixel art stworzoną w programie Pixel Studio. 8-bitowa mmuzyka została stworzona w darmowym programie beepbox.

5. **Wyniki:**
   Uzyskane punkty doświadczenia po śmierci bohatera mogą zostać zapisane w zewnętrznym pliku pdf. Od razu również można rozpocząć grę na nowo.

1.2 Historia aktualizacji

Wersja 0.2

- Stworzenie głównego pliku gry, dodanie ustawień, stworzenie testowej mapy, dodanie poruszania się i kolizji.
- Generowanie mapy (w tym stworzenie klas dla kafli, dla całej mapy oraz dla rysowania świata) oraz stworzenie testowej grafiki.

Wersja 0.4

- Przygotowanie mapy w programie tiled, stworzenie skryptu do czytania plików csv oraz dodanie pliku debug do testowania.
- Stworzenie grafiki dla potworów, postaci oraz otoczenia, Przygotowanie typów sprite'ów (niekolizyjnych oraz kolizyjnych), tworzenie mapy przez wczytywanie plików csv.

Wersja 0.6

- Uzyskiwanie punktów doświadczenia po pokonaniu wrogów.
- Dodanie menu kupna ulepszeń postaci za punkty doświadczenia.
- Dodanie wrogów, implementacja ataków postaci, stworzenie UI, oraz dodaniem interakcji z otoczeniem (niszczenie kwiatków, bycie przysłanianym przez trawę).

Wersja 0.8

- Dodanie ekranu śmierci, możliwości resetowania gry i zapisu wyników.
- Stworzenie i dodanie efektów dźwiękowych i muzyki.
- Ekran wygrania gry po zabiciu wszystkich wrogów.

Wersja 1.0

- Okna dialogowe podczas napotkania wroga.
- Elementy otoczenia urozmaicające grę (lawa zadaje obrażenia).
- Rozmieszczenie na mapie teleporterów.

1.3 Użyte technologie

1. Środowisko programistyczne - Python oraz moduły: Pygame i ReportLab
2. Pixel Studio - software użyty do stworzenia grafiki występującej w grze
3. BeepBox i BFXR - stworzenie całej ścieżki dźwiękowej
4. Tiled - rozplanowanie mapy i wygenerowanie plików CSV

2 Implementacja

Wyjaśnienie przykładowych mechanik

Cooldowny - po co i jak działają?

Porównajmy na chwilę sposób działania gry do filmu. Nieważne czy pełnoczy krótkometrażowy, film to inaczej zbiór zdjęć (klatek) aktualizowanych na tyle szybko by imitować płynny ruch. W przypadku gry, mamy oczywiście o wiele więcej róznych zmiennych wpływających na sam obraz jak np. input od gracza.

Nasz projekt działa w stałych 60 klatkach na sekundę, oznacza to że obraz aktualizowany jest 60 razy na każdą jedną sekundę. Biorąc to pod uwagę input od gracza staje się problemem, tj. pojedyncze wciśnięcie klawisza nie jest w stanie być na tyle szybkie by dotyczyło pojedynczej klatki. Gra w takim razie otrzymywałaby informacje od gracza nawet kilkadziesiąt razy po jednym wciśnięciu klawisza (np. 30 jeśli trwałoby 0,5 s). W zależności od funkcjonalności oznaczałoby to kilkadziesiąt ataków przy jednym wciśnięciu klawisza, kilkadziesiąt zapisów wyników czy też kilkadziesiąt zmian broni. Rozwiązaniem problemu będą tzw. timery oraz kilka prostych zmiennych.

Dla przejrzystości opiszemy działanie tylko jednego timera, konkretnie dla zapisywania wyników. Zaczynamy od utworzenia 2 zmiennych w funkcji "init"naszej klasy:

```
#stan pozwalający na zapisanie wyniku
self.switch_score = True

#pusty timer, który będzie odliczał czas od wciśnięcia klawisza
self.time_score = None
```

Dalej w funkcji ciągle sprawdzającej potencjalny input od gracza mamy kod:

```
#Jeśli gracz wciska "F" oraz stan zapisywania na to pozwala (ma wartość True)
if keys[pygame.K_f] and self.switch_score:

#przypisujemy datę do zmiennej
    now = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

#chwilowo zmieniamy stan pozwalający na zapisywanie na False
    self.switch_score = False

#wcześniej zadeklarowany timer zaczyna odliczać czas
    self.time_score = pygame.time.get_ticks()

#zapisujemy wynik wraz z datą do pliku tekstowego
with open("../list_of_scores.txt", "a") as file:
    file.write("Date: "+ str(now)+'\n' + "score: "+str(int(self.score))+'\n\n')
```

Niżej deklarujemy funkcję wszystkich czasów odnowienia. Oczywiście przedstawiamy skróconą wersję:

```
    def cooldowns(self):
#odliczamy aktualny czas
        current_time = pygame.time.get_ticks()

#jeśli stan pozwalający na zapisywanie ma wartość False
        if not self.switch_score:

#jeśli przy odjęciu aktualnego czasu od czasu liczonego od momentu wciśnięcia "F"
#otrzymamy wartość większą lub równą od 500 (losowa liczba, wymyślona na oko)
            if current_time - self.time_score >= 500:

#stan pozwalający na zapisywanie wraca do wartości True
                self.switch_weapon = True
```

Sama funkcja cooldowns wzywana jest w funkcji update, która jak nazwa sugeruje aktualizuje (w naszym konkretnym przypadku postać gracza)

```
    def update(self):
        self.cooldowns()
```

Sposób działania ofensywnej magii

Teraz przyjrzyjmy się działaniu funkcji odpowiedzilanej za działanie magii ofensywnej - zaklęcia które wysyła kulę energii we wszystkich kierunkach.

```
    class Spells:

#jednym z argumentów jest obiekt klasy tworzącej animacje
    def __init__(self,animation_maker):

# podajemy swojego rodzaju plakietkę dla czarów, ułatwia to później
#rozróżnianie ataków bronią od magii przy obliczaniu obrażeń zadanych wrogowi
        self.category = 'spells'
        self.animation_maker = animation_maker

        #przypisujemy do zmiennej plik dźwiękowy
        self.sound_energy_ball = pygame.mixer.Sound('../sound/ball.wav')

        #delikatnie ściszamy dźwięk
        self.sound_energy_ball.set_volume(0.2)
```

Jako argumenty przyjmujemy koszt energii, obiekt klasy gracz(żaba) oraz plakietki odnośnie rodzaju sprite'ów czyli wizualnej reprezentacji obiektów (pomaga to w łatwym przekazywaniu cech, np. attackable_sprites - sprite'y które można atakować i zniszczyć)

```
    def energy_ball(self,mana_cost,froggo,label):

#jeśli gracz posiada wystarczającą ilość energii
        if froggo.mana >= mana_cost:

#odejmujemy od energii gracza koszt za zaklęcie
            froggo.mana -= mana_cost
            for i in range(1,4):
```

```
9
10                  shift_plus= i*TILESIZE
11                  shift_minus= -i*TILESIZE
```

Następnie rysujemy 3 kule energii w każdym kierunku, dlatego odpowiednio dodajemy wcześniejsze zmienne kierunkowe do współrzędnych x,y naszej postaci

```
1 self.animation_maker.create_elements('energy_ball'(froggo.rectcenterx+shift_plus,
      froggo.rect.centery),label)
2                   self.animation_maker.create_elements('energy_ball'(froggo.rect.
                        centerx+shift_minus,froggo.rect.centery),label)
3                   self.animation_maker.create_elements('energy_ball',(froggo.rect.
                        centerx,froggo.rect.centery+shift_plus),label)
4                   self.animation_maker.create_elements('energy_ball'(froggo.rect.
                        centerx,froggo.rect.centery+shift_minus),label)
5                   self.animation_maker.create_elements('energy_ball'(froggo.
                        rectcenterx+shift_minus,froggo.rect.centery+shift_minus),label)
6                   self.animation_maker.create_elements('energy_ball'(froggo.rect.
                        centerx+shift_plus,froggo.rect.centery+shift_plus),label)
7                   self.animation_maker.create_elements('energy_ball'(froggo.rect.
                        centerx+shift_plus,froggo.rect.centery+shift_minus),label)
8                   self.animation_maker.create_elements('energy_ball'(froggo.rect.
                        centerx+shift_minus,froggo.rect.centery+shift_plus),label)
9               self.sound_energy_ball.play()
10              #przy każdym użyciu czaru odtwarzamy dźwięk
```

2.1 Klasy

Game:
Klasa przechowująca całą instancje programu, jest odpowiedzialna za tworzenie i inicjalizowanie okna gry.

World:
Przechowuje zmienne dotyczące mapy gry, takie jak grupy obiektów na niej przechowywanych (potwory, obiekty dekoracyjne, obiekty destrukcyjne), grafikę otoczenia i animacje.
Posiada funkcję "create world"używanej do tworzenia mapy poprzez czytanie wejściowych plików csv, oraz funkcje aktualizujące interfejs UI, jak "toggle menu"i "add exp".

Sprite:
Wszystkie obiekty wyświetlane na ekranie dziedziczą z klasy Sprite. Posiada zmienne związane z obiektem graficznym, takie jak jego pozycja na ekranie i ścieżka do pliku jpg/png wyświetlanego obrazka.
Posiada funckje pozwalające na wyświetlanie obiektów w oknie gry i odpowiednie ich grupowanie.

Tile:
Jedna "Płytka"mapy, dziedziczy z klasy Sprite, posiada konstruktor który pozwala w prosty sposób zmieniać cechy tworzonych płytek (na przykład ich wiel-

kość).

Froggo:
Klasa przechowująca wszystkie zmienne związane z postacią gracza (między innymi jego statystyki, dostępne zaklęcia, bronie).
Posiada funkcje odpowiedzialne za sterowanie postacią, wykrywanie kolizji, atakowanie i rzucanie zaklęć.

Monster:
Klasa przechowuje zmienne związane z potworem oraz jest odpowiedzialna za jego zachowanie (w tym wykrywanie obecności gracza, śledzenie go, atakowanie, wyświetlanie okien dialogowych).

Weapon:
Obiekt typu "broń", zawiera jego typ, statystyki, grafikę. Jest tworzony za każdym razem gdy gracz wykonuje atak. Wykrywane są kolizje między tym obiektem a wrogami.

Spell:
Klasa przechowująca funkcje definiujące zachowanie każdego z dostępnych zaklęć, jak na przykład "energy ball"i "heal".

UI:
Klasa stworzona z wielu funkcji odpowiedzialnych za rysowanie poszczególnych elementów interfejsu użytkownika na ekranie, takich jak "controls"wyświetlającą sterowanie oraz "bars"pokazującą pasek zdrowia i magii gracza.

Settings:
Posiada zmienne konfiguracyjne całej gry, na przykład wielkość okna i rodzaj używanej czcionki. Pozwala w łatwy sposób zmieniać ustawień gry globalnie.

2.2 Przechowywanie danych

Mapa:
Mapa gry jest przechowywana w plikach csv, gdzie każda płytka mapy ma przypisany numer. Po uruchomieniu gry, program chodzi po plikach csv i w zależności od wczytanej liczby generuje na tej pozycji odpowiedni obiekt.

Przykład:
W pliku map_Characters.csv zapisywane są startowe pozycje postaci, w tym wrogów oraz gracza. Szkielet jest oznaczany przez cyfrę 1, wilk przez 3, a brak żadnej postaci poprzez -1.

-1	1	-1	-1
-1	-1	-1	-1
-1	-1	3	-1
-1	-1	-1	-1

→

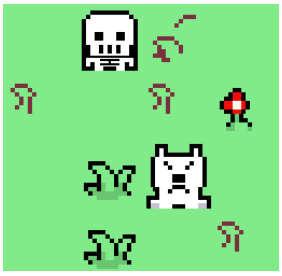
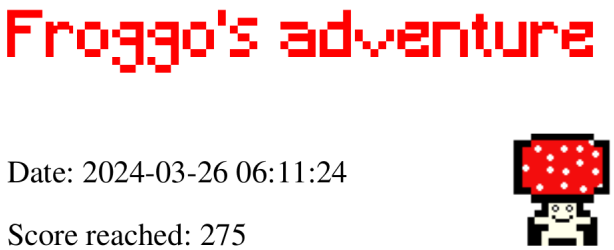


Tabela 1: Przykład części pliku csv i mapy wygenerowanej z niego, twórczość własna.

Zapis wyników:
Po utraceniu całego zdrowia gracz ma okazję zapisać swój wynik, będący wszystkimi punktami doświadczenia zyskanymi za pokonanie wrogów. Wynik jest zapisywany w pliku pdf w folderze "files"wraz z datą zdobycia określonego wyniku.
Przykład:



Certyfikat uzyskanego wyniku

3 Demonstracja

Uruchomienie gry
a) Uruchomienie terminala komand (cmd).
b) Upewnienie się, że moduły "pygame"oraz "reportlab"są zainstalowane. Można je zainstalować za pomocą komend "pip3 install pygame"oraz "pip3 install reportlab".
c) Korzystając z komendy "cd"przeniesienie się do folderu froggo_adventure a następnie files.
d) Użycie komendy "py main.py".

3.1 Instrukcje dla użytkownika

Sterowanie:

Poruszanie się postacią

Zmiana wybranej broni/magii

Atak bronią

Atak magią

Otwarcie menu ulepszeń postaci

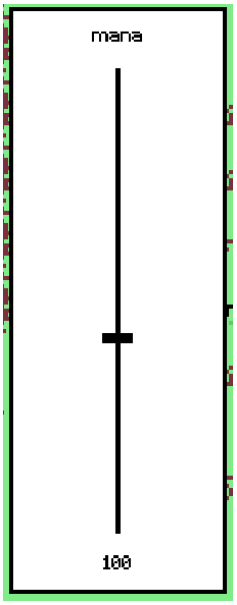
Restart gry

Interfejs użytkownika:

Pokazuje ilość aktualnego zdrowia

Pokazuje ilość punktów magii. odnawia się automatycznie

Pokazuje ilość punktów doświadczenia zyskanych za pokonanie wrogów.



Pozycja suwaka pokazuje poziom umiejętności postaci, ulepszenia można kupywać przyciskając strzałkę w górę.

Napis na górze suwaka reprezentuje nazwę ulepszenia, a liczba na dole pokazuje koszt jego kupna w punktach doświadczenia.

Przeciwnicy:

W grze znajdziemy trzy rodzaje przeciwników:


Groźny wilk!


Straszny szkielet!


Wściekły muchomor!

Po zobaczeniu gracza przeciwnicy zaczynają iść w jego kierunku, dopóki gracz nie odejdzie tak daleko, że stracą go z oczu. Kolizja z przeciwnikiem powoduje utratę punktów życia. Za pokonanie wrogów gracz zyskuje punkty doświadczenia których może użyć aby ulepszyć swoją postać.

Cel gry:

Celem gry jest pokonanie wszystkich przeciwników i zyskanie jak największego wyniku unikając przy tym spadku zdrowia do zera.
Aby pokonać wrogów na jego drodze, gracz ma do wyboru broń do walki wręcz oraz dwa zaklęcia - impuls energii i uzdrowienie. Używanie zaklęć kosztuje punkty magii, które odnawiają się automatycznie do swojego maksymalnego poziomu wraz z upływem czasu.

3.2 Drzewo folderowe

```
Froggo Adventure
├── files
│   ├── combat.py
│   ├── elements.py
│   ├── froggo.py
│   ├── main.py
│   ├── settings.py
│   ├── ui.py
│   ├── upgrade.py
│   └── world.py
├── img
│   ├── enemies
│   │   ├── bearDog
│   │   ├── mushroom
│   │   └── skeletor
│   ├── font
│   ├── objects
│   ├── player
│   └── weapons
├── map
│   ├── mapCharacters.csv
│   ├── mapDetails.csv
│   └── mapGround.csv
└── sound
    └── attacks
```

W celu zwiększenia czytelności drzewa, niektóre pliki i foldery zostały pominięte.
W korzeniu drzewa znajdują się cztery główne foldery:
files - przetrzymuje wszystkie pliki źródłowe gry.
img - zawiera grafikę i animacje wszystkich postaci w grze.
map - zawiera pliki csv, za pomocą których generowana jest mapa.
sound - zawiera muzykę i efekty dźwiękowe.

4 Wnioski i uwagi

Zakres pracy:

1. Część elementów graficznych zostało wykonanych przeze mnie szybciej w tym roku (przed ogłoszeniem szczegółów odnośnie projektu).

2. Głównymi inspiracjami były kursy na platformie YouTube z użyciem biblioteki PyGame (linki w bibliografii).

3. ChatGPT - pomoc w zakresie tworzenia wizualizacji systemu ulepszeń (podział na prostokąty które odpowiadają statystykom), całkowite stworzenie funkcji do tworzenia pdf-ów z wynikiem, ogólna pomoc z pytaniami odnośnie funkcjonalności różnych klas.

Trudności:

1. Sfera audiowizualna, mimo niskiej jakości sprawiła dość spory kłopot, głównie ze względu na kompletną subiektywność w ocenie rezultatów (stoi to w kontraście do części programistycznej, którą można uprościć do: działa poprawnie lub nie) a w związku z tym częste poprawki, usuwanie i ciągłe poczucie niezadowolenia i niedosytu.

2. W trakcie tworzenia gry doszło do nieudanych eksperymentów z próbą stworzenia alternatywnego sposobu sterowania postacią, konkretnie głosem. Gra miała korzystać z biblioteki SpeechRecognition ale doszło do kilku problemów:

    - Problem z wykryciem odpowiednich słów - w szczególności gdy istnieją podobne słowa, np. prawo i brawo. Przez to byłem zmuszony używać innych słów do komend ze względu na ich charakterystyczny wydźwięk co z drugiej strony sprawiało że lista komend była nieintuicyjna.
    - Zbyt wolne wykrycie - gra wymaga precyzji i refleksu czego komendy głosowe nie były w stanie zapewnić w takim samym stopniu jak sterowanie klawiaturą.
    - Konflikt programowy - przez specyfikę SpeechRecognition dochodziło do konfliktu z programem odpalającym grę, system nie chciał pozwolić by jednocześnie była odpalona wraz z mikrofonem. Przez to sama gra była albo całkowicie zapauzowana lub prędkość animacji z 60 klatek na sekundę zmieniała się na 1 klatka na 1 poprawną komendę.

3. Gra posiada kilka skrótów rozwiązaniowych (może nawet małych oszustw) by zapewnić trochę iluzję dopracowania.

    - Warunek zwycięstwa - możliwe zwycięstwo następuje gdy pokonani zostają wszyscy wrogowie, jednak warunek ten sprawdzany nie jest poprzez zeskanowanie mapy czy został na niej jakiś wróg. Wrogów

po prostu policzyłem i wprowadziłem twardy warunek na pokonanie konkretnie 13 wrogów. Oznacza to, że w przypadku losowej generacji wrogów gra nie miałaby poprawnego zwycięstwa.

- User Interface - gra działa w tej samej rozdzielczości - 1280x720. Sam interfejs niestety jest umieszczony w konkretnych miejscach na takim ekranie. Oznacza to że współrzędne to konkretne liczby a nie zmienne, które umożliwiłyby stałe przyklejene elementów do, np. spodu ekranu. Przez to jeśli doszłoby do zmiany rozdzielczości elementy interfejsu zostałyby wszystkie przesuniętę i porozrzucane po ekranie.

4. Skomplikowanie programu - przez pracę nad innymi projektami (w szkole jak i prywatnie) były okresy krótkich przerw od dopracowywania kodu gry. To w połączeniu z dość dużą ilością klas sprawiło że często zapominałem o zależnościach między klasami, które były współzależne od siebie. Wraz z co raz większym skomplikowaniem gry, niektóre kolejne funkcje były co raz bardziej ciężkie do implementacji.

4.1 Potencjalny rozwój

Projekt posiada szerokie możliwości rozwoju, pełna wersja gry mogłaby mieć więcej cech gier gatunku RPG, takich jak:

1. Fabuła
   Gra mogłaby posiadać historię, dającą podróży gracza głębszy cel.

2. Ekwipunek
   System ekwipunku, w którym mogłyby być umieszczane zdobyte bronie i zaklęcia oraz przedmioty upuszczane przez wrogów po ich śmierci.

3. Drzewa dialogowe
   Rozbudowanie systemu dialogów, pozwolenie graczowi na rozmowę z postaciami NPC. Mogłyby one również dawać graczowi zadania do wykonania za określoną nagrodę lub sprzedawać mu różne przedmioty.

4. Opcje dostosowania rozgrywki
   Dodanie menu główne gry, które umożliwiałoby zmienianie poziomu trudności rozgrywki oraz włączanie różnych opcji zwiększających dostępność (np. dostosowanie czcionki dla osób z dysleksją, alternatywne sterowanie, zwiększenie kontrastu w obrazie).

Literatura

[1] Dokumentacja modułu Pygame [https://www.pygame.org/docs; data dostępu: 28-05-2024]

[2] Dokumentacja języka Python [https://docs.python.org/pl/3; data dostępu: 28-05-2024]

[3] książka autorstwa Ala Sweigarta - Making Games with Python & Pygame

[4] Poradniki z Pythona i modułu Pygame [https://www.geeksforgeeks.org; data dostępu: 28-05-2024]

[5] Kanał z Poradnikami dotyczącymi Pygame [https://www.youtube.com/@ClearCode; data dostępu: 28-05-2024]

[6] Kurs Pygame [https://www.youtube.com/watch?v=tnq0whNwhZE&list=PLdBHMlEKo8UfnkIpJhwiOenHWxytitHpJ; data dostępu: 28-05-2024]

[7] Kurs Pygame [https://www.youtube.com/watch?v=i6xMBigpP4&list=PLzMcBGfZo4-lp3jAExUCewBfMx3UZFkh5; data dostępu: 28-05-2024]