Project Documentation

Zuzanna Brzóska

20.06.2020 r.

Spis treści

1. Wprowadzenie

Prezentowany projekt jest to gra inspirowana znaną grą The Sims. Jest to symulacja czynności życia codziennego kreowanych postaci. Gra jest w postaci tekstowej, gdzie gracz wybiera, co chce zrobić, będąc poniekąd na "pauzie" i instruując postacie w jakiej kolejności i jakie czynności mają wykonać, gdy gra ruszy.

Użytkowanie jest bardzo proste - wystarczy uruchomić plik "main" w projekcie, a następnie postępować zgodnie z poleceniami, wpisując odpowiednią liczbę odpowiadającą instrukcji, którą chcemy wybrać.

Na początku możemy zdecydować, czy chcemy zmienić coś w naszym domu, ponieważ możemy dodać w nim pomieszczenia takie jak łazienka (maksymalnie 3 na dom) i sypialnia (maksymalnie 4 na dom)

Później "wprowadzamy" do naszego domu członków rodziny, która będzie w nim mieszkać, nadając im imię, wiek oraz płeć. Rodzina może mieć maksymalnie

Następnie nadchodzi czas na grę, na tym etapie również możemy zdecydować o rozbudowie naszego domu lub powiększeniu rodziny. W tym miejscu zaczynamy zajmować się członkami rodziny, podnosząc poziom ich potrzeb takich jak: głód, energia, pęcherz, higiena, zabawa, komfort i towarzystwo. Maksymalna wartość to 10. Możemy również sprawić sobie zwierzątko do opieki, które należy karmić, a ono może nas podrapać lub przytulić właściciela wpływając na jego potrzeby.

2. Architektura programu

Projekt składa się z siedmiu klas reprezentujących dom, pokoje, postaci oraz zwierzęta, a także z symulacji gry, która pozwala w łatwy sposób korzystać z gry bez znajomości języka programowania Python. Dołączone są również testy jednostkowe klas.Dom oraz postać są głównymi klasami tego programu, można tę grę porównać do gry planszowej, wtedy dom pełni rolę planszy a postacie rolę pionków.

Pokoje są dość podobnie zbudowane, jednak są osobnymi klasami, ponieważ sposób korzystania z różnych pomieszczeń różni się m.in. czasem korzystania lub pojemnością pomieszczeń. Na przykład zjedzenie przekąski zabiera jeden krok czasu, podczas gdy spanie zabiera ich trzy. Podobnie z łazienki może korzystać tylko jedna osoba naraz, mimo że można wykonać tam dwie czynności, a z salonu mogą korzystać wszyscy naraz oglądając telewizję wspólnie.

Projekt składa się z klas:

a. House
b. Character
c. Pet
d. Bathroom
e. Bedroom
f. Kitchen
g. Livingroom

3. Klasa House

Jak wspomniałam wcześniej, klasa House pełni poniekąd funkcję planszy. Gra odbywa się w obrębie jednego domu, postacie i zwierzęta poruszają się po nim, pokoje są częścią domu.

Atrybutami tej klasy są: rooms (pamięta pokoje które składają się na dom), family (słownik, gdzie kluczami są imiona, a wartościami członkowie rodziny), time (odlicza czas trwania symulacji) oraz address (dodatkowa informacja).

Wśród metod tej klasy znajduje się jedna z najważniejszych funkcji mojego projektu czyli metoda run(steps), która wywołana poniekąd odpowiada puszczeniu “play” w grze, czyli dzięki niej postacie wykonują czynności, które użytkownik dla nich zaplanował, na tyle kroków czasu ile podamy.

Drugą ważną metodą jest move_in_sim(name, age, sex), która odpowiada za wprowadzenie nowego członka rodziny do domu. Poza tym metodami klasy House można również dodać łazienkę, sypialnię oraz sprawdzić potrzeby wszystkich członków rodziny naraz.

4. Klasa Character

Jest to najbardziej rozbudowana klasa tego projektu, pełni funkcję podobną do "pionków" w grze planszowej. Jest to element projektu, który korzysta z pozostałych elementów i którym gracz może sterować.

Atrybutami tej klasy są: name, age, sex, pet, house (informacyjne), needs (słownik, gdzie klucze to nazwy potrzeb, a wartości to poziom potrzeby), current_activity (są czynności które trwają dłużej niż jeden krok czasu, takie jak spanie i gotowanie, więc zapisuję tę informację, by postać nie znajdowała się w dwóch miejscach naraz), activity_queue (kolejka czynności wybranych przez gracza do wykonania).

Klasa Character posiada bardzo dużo metod, ponieważ za każdą czynność odpowiadają dwie metody: jedna do zapisania czynności w kolejce do wykonania później w trakcie gry. Jest ich za dużo by wszystkie wypisać, więc można je znaleźć na diagramie UML.

Większość metod odpowiadających za czynności w grze zwyczajnie wpisują postać do kolejek w pomieszczeniach (lub wykonują się od razu jak nakarmienie zwierzęcia), jednak use_toliet(), use_shower() oraz use_bed() jeszcze najpierw sprawdzają, czy w domu znajduje się więcej łazienek/sypialni niż jedna, a jeżeli tak, to wybierają tę, w której kolejka jest krótsza. Natomiast talk(friend) wpływa nie tylko na potrzebę "social" postaci, która ją wykonuje, ale również na potrzebę postaci, która jest argumentem tej metody.

Poza metodami odpowiadającymi za czynności postaci, są również inne pomagające w rozgrywce. Przede wszystkim check_queue() rozładowuje kolejkę czynności do wykonania, co pozwala postaci przenosić się do pomieszczeń oraz needs_down(), które wykonuje się co każdy krok w symulacji i obniża poziom potrzeb postaci wraz z upływającym czasem. Oprócz tego show_info(), check_needs(), check_queue() przydają się do sprawdzenia informacji nt. postaci w trakcie rozgrywki. Poza tym jeśli popełnimy błąd lub zmienimy zdanie w trakcie wybierania czynności dla postaci, możemy je usunąć i zacząć od nowa dzięki empty_queue(), możemy sprawić postaci zwierzę za pomocą get_pet(name). Nie musimy się również martwić, jeśli któraś z najważniejszych potrzeb potrzebnych do przeżycia (głód i energia) spadnie za bardzo, ponieważ metoda raise_alert() ostrzeże nas przed tym oraz automatycznie doda metody sleep()/eat_snack() na początek kolejki czynności.

5. Klasa Pet

Zwierzę jest przypisane do jednej postaci, ponieważ każde zwierzę musi mieć swojego właściciela w domu. Jest ona dość krótka, ponieważ nie możemy sterować zwierzęciem. a jedyną jego potrzebą jest głód. Jedyne atrybuty tej klasy to: name, owner i hunger (informacyjne).

Metody hunger_down() oraz alert() są analogiczne do metod needs_down() i raise_alert() odpowiednio w klasie Character. Ostatnia metoda act() odpowiada za to czy zwierzę wejdzie w interakcję z właścicielem - w metodzie run() w klasie House losujemy czy zostanie wywołana ta metoda, a następnie w act() losujemy czy zwierzę zachowa się pozytywnie czy negatywnie - i tak też wpłynie na potrzeby właściciela. Gdy zwierzę jest głodne (poziom hunger poniżej 4/10), wzrasta prawdopodobieństwo, że zareaguje negatywnie.

6. Klasy pokoi

Pozostałe klasy reprezentujące pokoje w domu opiszę wspólnie, ponieważ są dość podobne oraz krótkie. Każda z klas Bathroom, Bedroom, Livingroom i Kitchen posiada jako atrybuty pewne kolejki, a także metodę check_queue(), która te kolejki rozładowuje i wywołuje kolejne metody.

Klasa Bathroom posiada tylko jedną kolejkę (atrybut queue), ponieważ tylko jedna osoba naraz może korzystać z tego pomieszczenia, dlatego też w tej kolejce zapisana jest nie tylko osoba ale wraz z nią czynność, którą chce ona wykonać. Metody use_bathroom() i use_toilet() zwyczajnie podnoszą poziom potrzeb ‘hygiene’ oraz ‘bladder’ u postaci do poziomu 10.

Klasa Bedroom również posiada tylko jedną kolejkę, ponieważ w sypialni można skorzystać tylko z łóżka. Ten pokój różni się od pozostałych również tym, że można wybrać wielkość łóżka (1 lub 2) tym samym zmieniając pojemność pokoju. Dodatkowo posiada atrybut current_users, gdzie zapisane są postaci aktualnie korzystające z łóżka, ponieważ czynność spania zajmuje 3 kroki czasu. Metoda use_bed() podnosi poziom potrzeby ‘energy’, podczas spania o 1 w górę, a na zakończenie do 10.

Klasa Kitchen posiada dwie kolejki: do lodówki (fridge_queue) oraz kuchenki (oven_queue), a także atrybuyt current_oven_user, bo gotowanie zajmuje dwa kroki czasu. Use_fridge() i use_oven() podnoszą potrzebę ‘hunger’ - use_fridge() o 2, a use_oven() do 10 na zakończenie gotowania.

Klasa Livingroom posiada jedną kolejkę do kanapy (couch_queue). Posiada również atrybuty: current_couch_users, bo są 3 miejsca na kanapie, oraz current_tv_users() ponieważ do używania telewizora nie potrzeba kolejki,

dlatego że korzystać z niego mogą wszyscy naraz. Metody use_couch() i use_tv() podnoszą potrzeby ‘comfort’ i ‘fun’ do poziomu 10.

7. Diagram UML

Poniżej prezentuję diagram UML mojego projektu: