**PROJEKT ZALICZENIOWY – PROGRAMOWANIE OBIEKTOWE II (Python)**

Julia Kulczycka

**Skrócony opis i cel gry**

Postać gracza porusza się po prostokątnej planszy w czterech kierunkach. Plansza na początku jest zakryta – gracz poruszając się odkrywa kolejne pola, na których może znaleźć panaceum, broń, litery hasła lub spotkać wrogów. Celem gry jest skompletowanie wystarczającej ilości liter, aby zgadnąć hasło (wszystkie hasła są po angielsku, związane ze zwierzętami domowymi).  
Grę rozpoczynamy uruchamiając plik Game.py

**Spis klas (razem z podklasami)**

1. Board
2. Player
   * Rat
   * Cat
   * Dog
3. Key
4. Item
   * Weapon
   * Panacea
   * Letter
   * Enemy

**Opis architektury systemu**

1. Inicjowanie planszy

Aby rozpocząć grę, musimy stworzyć planszę (*Board*) – w tym celu podajemy programowi żądaną przez nas długość planszy (*Board.l*) oraz wysokość (*Board.h*). Jeżeli podamy liczbę mniejszą od 5, program automatycznie zmieni ją na 5 i wypisze komunikat *'Minimal length/height is 5'*.

Podczas konstruowania planszy w sposób losowy niektórym polom zostaną przypisane „znajdujące się na nich” przedmioty (*Item*): panaceum (*Panacea*), broń (*Weapon*) oraz wrogowie (*Enemy*). Ich ilość jest zależna od pola (*Board.l \* Board.h*) planszy – ilość wrogów wynosi , a ilość broni i panaceum . Utworzone zostają słowniki:  
*Board.meds* – kluczami słownika są krotki zawierające współrzędne, na których znajduje się panaceum, wartościami są obiekty klasy *Panacea*  
*Board.weapons* – kluczami słownika są krotki zawierające współrzędne, na których znajdują się bronie, wartościami są obiekty klasy *Weapon  
Board.enemies* - kluczami słownika są krotki zawierające współrzędne, na których w chwili początkowej znajduje się wróg, wartościami są obiekty klasy *Enemy*  
Może się zdarzyć, że kilka przedmiotów różnych podklas (a więc – znajdujących się w różnych słownikach) znajdzie się na jednym polu.

1. Losowanie hasła i umieszczenie liter na planszy

Po utworzeniu planszy, z pliku tekstowego (*keys.txt*) wylosowane zostaje hasło. Jego litery (*Letter.letter*) są wartościami słownika *letters*, którego kluczami są krotki ze współrzędnymi na planszy poszczególnych liter.

1. Tworzenie postaci gracza

Po stworzeniu planszy i umieszczeniu na niej wszystkich przedmiotów (podklasy *Item*), gracz może utworzyć swoją postać (*Player*). Istnieją 3 podklasy – *Rat, Cat, Dog* - spośród których gracz wybiera jedną, którą będzie grać.

Każda postać posiada początkową ilość punktów zdrowia (*Player.hp*) oraz siły ataku (*Player.att*). Dodatkowo dla każdej podklasy zdefiniowany jest atrybut *max\_hp*, który określa maksymalną ilość punktów zdrowia, którą może posiadać gracz.

Każda postać może poruszać się w górę (*Player.move\_up*), w dół (*Player.move\_down*), w prawo (*Player.move\_right*)   
i w lewo (*Player.move\_left*) po planszy (nie ma możliwości wyjścia poza planszę). Może również brać udział w walce z wrogiem używając metody *Player.attack(enemy)* oraz jeden raz w każdej rozgrywce użyć swojej supermocy *Player.**superpower* (superpower jest metodą abstrakcyjną, nadpisywaną w podklasach).

Podklasy postaci różnią się wartościami niektórych atrybutów, supermocą i wyglądem:

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Rat* | *Cat* | *Dog* |
| \_\_str\_\_ | 🐀 | 🐈‍ | 🐕 |
| hp | 150 | 100 | 70 |
| att | 10 | 25 | 50 |
| max\_hp | 150 | 100 | 70 |
| superpower | **Sewer teleport**  Gracz może przeteleportować się w dowolnie wybrane pole na planszy | **Cat eye**  Gracz może zobaczyć aktualne lokalizacje wszystkich wrogów | **Super sniff**  Gracz może wyszukać na planszy wszystkie bronie lub wszystkie zasoby panaceum |

(Wygląd gracza jest dobrze wyświetlany w konsoli PyCharma… w wierszu poleceń może się pokazywać ▯)

Postać gracza w chwili początkowej znajduje się na środku planszy (*self.x = self.board.l // 2, self.y = self.board.h // 2*).

1. Główna pętla gry

Pętla wykonuje się, dopóki gracz ma więcej niż 0 punktów życia (*player.hp*) oraz prób zgadnięcia hasła (*Key.shots*).

Każda iteracja pętli zaczyna się od wyboru gracza poprzez podanie do wejścia:  
A/a – ruch w lewo (*Player.move\_left*)  
W/w – ruch w górę (*Player.move\_up*)  
S/s – ruch w dół (*Player.move\_down*)  
D/d – ruch w prawo (*Player.move\_right*)  
Q/q – użycie supermocy (*Player.superpower*)  
1 – próba zgadnięcia hasła (*Key.guess*)  
0 – wyjście z gry  
W przypadku podania innego znaku, program informuje o problemie i musimy po raz kolejny dokonać wyboru.

Jeżeli gracz wybrał ruch, który jest możliwy, wykonuje go (w przeciwnym razie zaczyna się kolejna iteracja). Następnie w słownikach *letters*, *Board.enemies*, *Board.meds*, *Board.weapons* sprawdzane są klucze. Jeżeli krotka współrzędnych gracza (*(Player.x, Player.y)*) znajduje się w poszczególnych słownikach, wykonywane są akcje:  
*Board.enemies* – wróg atakuje (*Enemy.attack(player)*), po czym uruchamia się pętla walki (szczegółowo opisana w punkcie 5)  
*Board.meds* – jeżeli na polu nie znajduje się wróg, zostaje zebrane znajdujące się na nim panaceum i ulecza gracza (*Panacea.heal\_player(player)*) *Board.weapons* – jeżeli na polu nie znajduje się wróg, zostaje zebrana znajdująca się na nim broń i dodaje graczowi pewną ilość punktów ataku (*Weapon.collect\_weapon(player)*)  
*letters* – jeżeli na polu nie znajduje się wróg, litera jest znaleziona (*Key.decode*) i pojawia się w haśle (*Key.encoded*)

Po wykonaniu ruchu pole, na którym poprzednio stała postać gracza (o współrzędnych *(player.px, player.py)*) zmienia swój wygląd na planszy na pustą kratkę. Dzięki temu gracz wie, które pola już odwiedził, a które nie. Wyjątek stanowi sytuacja, gdy gracz spotkał wroga, ale go nie pokonał – pole wroga zostaje oznaczone jako „x”.

Jeżeli gracz wybrał „Q” lub „q”, wykonuje się metoda *Player.superpower* przypisana podklasie jego postaci (poszczególne supermoce zostały dokładniej opisane w charakterystyce klas)

Jeżeli gracz wybrał „1”, musi podać do wejścia słowo, które jego zdaniem, jest hasłem (*Key.key*). Jeżeli będzie zgodne, wypisany zostaje komunikat *„YOU WON!”* i pętla gry się przerywa. Jeżeli wpisane hasło nie będzie poprawne, wypisuje się komunikat „*No, it’s not the key*” i liczba prób zgadnięcia hasła (*Key.shots*) zmniejsza się o jeden. Jeżeli liczba prób jest równa zero, pętla gry również się przerywa.

Jeżeli gracz wybrał „0”, musi potwierdzić, że chce wyjść z gry. Jeśli potwierdzi, pętla gry się przerywa. W przeciwnym wypadku rozpoczyna się następna iteracja.

Każdy obrót głównej pętli gry kończy się na ruchu wrogów (*Board.update*).

1. Pętla walki

Jeżeli gracz znajdzie się na tym samym polu, co jeden z wrogów (tzn. krotka współrzędnych postaci gracza *(Player.x, Player.y)* znajdzie się wśród kluczy słownika *Board.enemies*), wróg przypisany do tych współrzędnych atakuje gracza  
(*Enemy.attack(player)*). Następnie uruchamia się pętla walki, która wykonuje się, dopóki gracz i wróg mają więcej niż 0 punktów życia.

Każdy obrót pętli walki rozpoczyna się od wyboru gracza. Jeśli do wejścia poda „0”, oznacza to, że wybiera ucieczkę. Pętla walki się wtedy zamyka, pole, na którym „znajdował się” wróg zostaje oznaczone na planszy (*str(Board)*) jako „x” i rozpoczyna się kolejna iteracja głównej pętli gry.  
Jeżeli natomiast gracz wybierze „1”, oznacza to, że chce kontynuować walkę. Wtedy najpierw gracz atakuje wroga (*Player.attack(enemy)*), a następnie wróg atakuje gracza (*Enemy.attack(player)*). Potem, o ile obydwoje mają więcej niż 0 punktów życia, zaczyna się kolejna iteracja pętli walki.

1. Zakończenie gry

Gra może skończyć się trzema wynikami:  
Wygrana – gracz podał poprawne hasło  
Przegrana – ilość punktów życia gracza (*Player.hp*) spadła poniżej 1 lub nie ma więcej prób odgadnięcia   
hasła (*Key.shots*)  
Brak wyniku – gracz wybrał wyjście z gry (wybór „0” na początku iteracji głównej pętli)

**Opis klas**

1. Board – plansza, główna klasa odpowiedzialna za działanie gry

Atrybuty instancji:

1. *self.l: int* – długość planszy tzn. długość list będących elementami listy *self.table*
2. *self.h: int* – wysokość planszy tzn. ilość elementów listy *self.table*
3. *self.table: list* – lista *self.h* list o długości *self.l*, których elementy będą polami planszy
4. *self.meds: dict* – kluczami słownika są krotki zawierające współrzędne, na których znajduje się panaceum, wartościami są obiekty klasy *Panacea*
5. *self.weapons: dict* – kluczami słownika są krotki zawierające współrzędne, na których znajdują się bronie, wartościami są obiekty klasy *Weapon*
6. *self.enemies: dict* – kluczami słownika są krotki zawierające współrzędne, na których w chwili początkowej znajduje się wróg, wartościami są obiekty klasy *Enemy*

Metody:

1. *\_\_init\_\_(self, l, h)* – konstruktor został już właściwie opisany w punkcie 1 **opisu architektury systemu**.
2. *\_\_str\_\_(self)* – wypisanie planszy jako PrettyTable (klasa z biblioteki prettytable).
3. *move\_player(self, player, k = ‘ ‘)* – ruch postaci gracza na planszy. Postać gracza pojawia się na polu w *self.table[player.y][player.x]*. Pole, które uprzednio zajmował gracz, zostaje oznaczone w *self.table* jako *k*.
4. *move\_enemies(self)* – “aktualizacja” (a właściwie utworzenie od nowa) słownika *self.enemies* z aktualnymi współrzędnymi wrogów.

2. Item – klasa przedmiotu, mająca 4 podklasy: Panacea, Weapon, Letter, Enemy

Atrybuty instancji:

1. *self.board: Board* – plansza, na której istnieje przedmiot
2. *self.x: int* – pierwsza współrzędna lokalizacji przedmiotu na planszy
3. *self.y: int* – druga współrzędna lokalizacji przedmiotu na planszy

Metody:

1. *\_\_init\_\_(self, board)* – przypisuje planszę *board* do self.board, następnie losuje wartości *self.x* oraz *self.y* spośród liczb całkowitych odpowiednio z przedziałów [0, *self.board.l*) oraz [0, *self.board.h*)

3. Weapon(Item) – broń, która zwiększa siłę ataku gracza

Atrybuty klasy:

1. *d: dict* – słownik zawierający nazwy (w kluczach) i siły ataku (w wartościach) poszczególnych rodzajów broni

Atrybuty instancji:

1. *self.board, self.x, self.y* odziedziczone z klasy-rodzica Item
2. *self.name: str* – nazwa broni
3. *self.force: int* – siła broni (wartość, która zostaje dodana do siły ataku gracza, gdy znajdzie daną broń)

Metody:

1. *\_\_init\_\_(self, board)* – oprócz konstruktora odziedziczonego z klasy-rodzica, spośród kluczy słownika *d* zostaje wylosowana nazwa broni *self.name* po czym do *self.force* zostaje przypisana siła tej broni (wartość w słowniku *d* dla klucza *self.name*)
2. *collect\_weapon(self, player)* – podniesienie/znalezienie broni przez gracza *player*; wypisanie komunikatu o znalezieniu broni i dodanie siły ataku broni *self.force* do siły ataku gracza *player.att*

4. Panacea(Item) – panaceum, który uleczy gracza

Atrybuty instancji:

1. *self.board, self.x, self.y* odziedziczone z klasy-rodzica Item
2. *self.heal: int* – siła leczenia (wartość, która zostaje dodana do punktów życia gracza, gdy znajdzie panaceum)

Metody:

1. *\_\_init\_\_(self, board)* – oprócz konstruktora odziedziczonego z klasy-rodzica, wartości *self.heal* zostaje przypisana losowo wybrana liczba całkowita z przedziału [10, 50)
2. *heal\_player(self, player)* – uleczenie gracza *player*; dodanie do *player.hp* wartości *self.heal* lub maksymalnej ilości punktów życia możliwej dla typu gracze *type(player).max\_hp*, jeżeli suma by ją przekroczyła

5. Letter(Item) – litera hasła na planszy

Atrybuty instancji:

1. *self.board, self.x, self.y* odziedziczone z klasy-rodzica Item
2. *self.letter: str* – litera

Metody:

1. *\_\_str\_\_* – litera jako string

6. Enemy(Item) – wróg, który może zaatakować gracza

Atrybuty klasy:

1. *d: dict* – słownik zawierający w kluczach nazwy, a w wartościach krotki (punkty życia, siła ataku) poszczególnych wrogów

Atrybuty instancji:

1. *self.board, self.x, self.y* odziedziczone z klasy-rodzica Item
2. *self.name: str* – nazwa wroga
3. *self.hp: int* – punkty życia wroga
4. *self.att: int* – siła ataku wroga (tyle *hp* zabiera graczowi po udanym ataku)

Metody:

1. *\_\_init\_\_(self, board)* – oprócz konstruktora odziedziczonego z klasy-rodzica, spośród kluczy słownika *d* zostaje wylosowana nazwa wroga *self.name* po czym do *self.hp* oraz *self.att* zostają przypisane odpowiednio zerowy i pierwszy element krotki, będącej wartością w słowniku *d* dla klucza *self.name*
2. *attack(self, player)* – zaatakowanie gracza *player*; szansa na udany atak wynosi . Jeżeli atak jest udany, *player* traci tyle punktów życia, ile wróg ma siły ataku i wypisany jest komunikat o udanym ataku. W przeciwnym razie wypisany zostaje komunikat, że atak wroga się nie udał
3. *move(self)* – ruch wroga tzn. zmienienie jego współrzędnych *self.x* oraz *self.y*; wrogowie poruszają się w sposób losowy we wszystkich kierunkach (również na ukos) zawsze o jedno pole

6. Key – hasło, którego zgadnięcie jest celem gracza

Atrybuty instancji:

1. *self.key: str* – hasło, które musi zgadnąć gracz
2. *self.encoded: list* – aktualnie znane przez gracza litery hasła
3. *self*.*shots: int* – pozostałe próby zgadnięcia hasła przez gracza

Metody:

1. *\_\_init\_\_(self, board)* – z pliku tekstowego *key.txt* zostaje wylosowane hasło i przypisane wartości *self.key*. *self.encoded* w chwili początkowej jest listą tylu znaków „\_”, ile liter jest w haśle. Do *self.shots* przypisana zostaje liczba 3
2. *\_\_str\_\_(self)* – string reprezentujący aktualnie znane litery hasła tzn. *self.encoded*
3. *decode(self, letter)* – odszyfrowanie jednej literki w *self.encoded*

7. Player – klasa gracza, mająca trzy podklasy: Rat, Cat, Dog

Atrybuty instancji:

1. *self.board: Board* – plansza, na której istnieje gracz
2. *self.x: int* – pierwsza współrzędna aktualnej lokalizacji gracza
3. *self.y: int* – druga współrzędna aktualnej lokalizacji gracza
4. *self.px: int* – pierwsza współrzędna poprzedniej lokalizacji gracza (przed ostatnim ruchem)
5. *self.py: int* – druga współrzędna poprzedniej lokalizacji gracza (przed ostatnim ruchem)
6. *self.hp: int* – ilość punktów życia gracza
7. *self.att: int* – wartość siły ataku gracza
8. *self.sp: bool* – wartość boolowska „pamiętająca”, czy gracz wykorzystał już swoją supermoc

Metody:  
Ruch jest możliwy, jeśli nie spowoduje wyjścia gracza poza planszę, tzn. „wypadnięcia” wartości *self.x* z przedziału   
[0; *self.board.l*), a wartości *self.y* z przedziału [0; *self.board.h*)

1. *\_\_init\_\_(self, board)* – konstruktor klasy. *self.board* zostaje przypisana *board*. Na początku gracz znajduje się na środku planszy (na tyle, na ile to możliwe), stąd *self.x = self.board.l // 2* oraz *self.y = self.board.h // 2*. Początkowa wartość *self.sp* to *False*. *self.hp* oraz *self.att* zostaną przypisane w podklasach
2. *move\_up(self)* – ruch do góry, o ile jest możliwy; przypisanie *self.y* do *self.py* i odjęcie 1 od *self.y*
3. *move\_down(self)* – ruch w dół, o ile jest możliwy; przypisanie *self.y* do *self.py* i dodanie 1 do *self.y*
4. *move\_right(self)* – ruch w prawo, o ile jest możliwy; przypisanie *self.x*do *self.px*i dodanie 1 do *self.x*
5. *move\_left(self)* – ruch w lewo, o ile jest możliwy; przypisanie *self.x* do *self.px* i odjęcie 1 od *self.x*
6. *attack(self, enemy)* – zaatakowanie wroga przez gracza; szansa na udany atak wynosi . Jeżeli atak jest udany, *enemy* traci tyle punktów życia, ile gracz ma siły ataku i wypisany jest komunikat o udanym ataku. W przeciwnym razie wypisany zostaje komunikat, że atak gracza się nie udał
7. *superpower(self)* – supermoc; metoda abstrakcyjna, która jest nadpisana w podklasach. Gdy gracz wykorzysta swoją supermoc, wartość *self.sp* zmiania się na *True*

7. Rat(Player) – niezbyt silny, ale niezwykle wytrzymały szczur

Atrybuty klasy:

1. *max\_hp = 150* – maksymalna ilość punktów życia, które może posiadać gracz

Atrybuty instancji:

1. Wszystkieodziedziczone z klasy-rodzica Player

Metody:

1. *\_\_init\_\_(self, board)* – konstruktor klasy rodzica; przypisuje *self.hp = 150* oraz *self.att* = 10
2. *\_\_str\_\_(self)* – emotikona 🐀
3. *superpower(self)* – supermocą klasy *Rat* jest możliwość przeniesienia się na dowolnie wybrane pole. Wypisany zostaje komunikat o tym, że supermoc została aktywowana, po czym gracz musi podać kolejno współrzędne *x* i *y* na które chce się przenieść. Sprawdzana jest poprawność każdej współrzędnej i jeżeli zostaje zatwierdzona, *x* i *y* zostają przypisane odpowiednio do *self.x* oraz *self.y*

8. Cat(Player) – najbardziej zbalansowany zwierzak, czyli kot

Atrybuty klasy:

1. *max\_hp = 100* – maksymalna ilość punktów życia, które może posiadać gracz

Atrybuty instancji:

1. Wszystkieodziedziczone z klasy-rodzica Player

Metody:

1. *\_\_init\_\_(self, board)* – konstruktor klasy rodzica; przypisuje *self.hp = 100* oraz *self.att* = 25
2. *\_\_str\_\_(self)* – emotikona 🐈‍
3. *superpower(self)* – supermocą klasy *Cat* jest pokazanie aktualnego położenia wszystkich wrogów na planszy. Wypisany zostaje komunikat o tym, że supermoc została aktywowana, po czym uruchamiają się zagnieżdżone iteracje po kolejnych polach planszy. Jeżeli na jakimś polu znajduje się wróg (współrzędne pola znajdują się w kluczach słownika *self.board.enemies*), zostaje ono oznaczone jako „x”. Jeżeli na którymś z pól, które wcześniej zostało oznaczone jako „x” nie ma już wroga, zostaje ono oznaczone jako nieodkryte („?”)

9. Dog(Player) – bardzo silny, ale niezbyt wytrzymały pies

Atrybuty klasy:

1. *max\_hp = 70* – maksymalna ilość punktów życia, które może posiadać gracz

Atrybuty instancji:

1. Wszystkieodziedziczone z klasy-rodzica Player

Metody:

1. *\_\_init\_\_(self, board)* – konstruktor klasy rodzica; przypisuje *self.hp = 70* oraz *self.att* = 50
2. *\_\_str\_\_(self)* – emotikona 🐕
3. *superpower(self)* – supermocą klasy *Dog* jest pokazanie wszystkich znajdujących się na planszy broni lub panaceum. Wypisany zostaje komunikat o tym, że supermoc została aktywowana, po czym gracz dokonuje wyboru, czy chce „wywęszyć” panaceum, czy broń. Następnie uruchamiają się zagnieżdżone iteracje po kolejnych polach planszy. Jeżeli na jakimś polu znajduje się szukany przedmiot (współrzędne pola znajdują się w kluczach słownika *self.board.meds* lub *self.board.weapons*), zostaje ono oznaczone jako „O”

**Diagram UML**

