# Ťahák 2

### Triedy a objekty

Často sa nám pri návrhu hry môže stať, že potrebujeme viacero vecí, ktoré vyzerajú rovnako, správajú sa rovnako a líšia sa napríklad len pozíciou. Naviac ale každá z týchto vecí chce existovať samostatne, mimo ostatných podobných vecí. Na docielenie práve tohoto dizajnu nám pomáha objektovo orientovaný návrh.

Ak potrebujeme používať viacero rovnakých vecí, môžeme si navrhnúť šablónu, ktorá bude popisovať ako sa dané veci správajú. Túto šablónu nazveme `trieda`. Trieda má dve dôležité časti: `vlastnosti`, takže premenné, ktoré popisujú danú vec (napríklad pozícia vo svete, HP, ktoré pridá) a `metódy`. Metódy sú funkcie, ktoré daná vec vykonáva nad svojimi vlastnosťami. Napríklad môžeme mať metódu na pohyb, ktorá posunie príslušnú vec.

Trieda je však len šablóna, ktorá hovorí ako vieme vytvoriť konkrétnu vec. Z triedy vieme vytvárať jednotlivé `objekty`, čo sú konkrétne veci v programe. Tie majú svoje vlastné hodnoty vo vlastnostiach a sú nezávislé od ostatných objektov.

Ďalej si uvedomme, že ak chceme pristupovať k nejakej vlastnosti alebo metóde, tak tieto vlastnosti **nepatria triede**, ale **patria objektu**. Môžeme teda pristupovať len ku vlastnostiam daného objektu. Tento objekt má svoje meno a keď chceme pristúpiť k jeho vlastnosti alebo chceme nad ním vykonať metódu, použijeme `.`(bodka), ktorá spája objekt s jeho vlastnoťou.

Navyše, ak sa chceme v rámci metódy odkazovať na vlastnosti daného objektu, použijeme pomocné slovíčko `self`. Takisto nezabudnite, že premenné, ktoré má mať daná trieda ako vlastnosti, musia byť uvedené v špeciálnej funkcii, ktorá sa volá konštruktor. V našom prípade bude mať táto funkcia názov `nastav()`.

Dôležité je ešte si povedať, že ak chceme vytvoriť nový objekt pre triedu s názvom `Trieda`, vytvoríme ho tak, že zavoláme funkciu `Trieda()`. Táto funkcia nemusí byť implementovaná, je to špeciálne značenie, ktoré na pozadí zavolá funkciu `nastav()`.

### Trieda v Gaminatore

Ak chceme vytvoriť triedu v Gaminatore, potrebujeme vedieť niekoľko základných vlastností a metód, ktoré takáto trieda musí mať. V prvom rade, musí mať každý objekt svoju pozíciu v našom svete. Preto mu musíme nastaviť vlastnosti $x$ a $y$. Bod $(x, y)$ sa dá následne chápať ako stred našeho objektu.

Najdôležitejšie sú však metódy, ktoré musíme implementovať. Myšlienka celého cyklu triedy je jednoduchá. Na začiatku objekt **vznikne** a nastaví si začiatočné vlastnosti. Následne sa bude v **každom kroku** programu vykonávať nejaká zmena tohoto objektu (napr. ryba pláve, jedlo padá ...) a toto pokračuje, až kým nám je objekt zbytočný a chceme ho **odstrániť**.

Pri vytvorení objektu sa volá funkcia `nastav(self)`. Do tejto metódy je potrebné napísať, aké vlastnosti má daná trieda a aké sú ich začiatočné hodnoty, keď sa bude vytvárať nový objekt. Počas behu programu sa potom bude volať metóda `krok(self)`, do ktorej chceme napísať čo má urobiť daný objekt v každom kroku programu. No a na konci treba zavolať metódu `znic()`, ktorá zničí daný objekt. Túto funkciu neprogramujete vy, je už vopred pripravená.

```
class Ryba(Vec):
    def nastav(self):
        self.x = 10
        self.y = 10

    def krok(self):
        self.y += 1
```

### Kreslenie objektov

Objekty sa musia nejakým spôsobom vykresľovať. Na to budeme opäť používať `kreslic`. Ako však vieme, každý objekt má nejakú svoju pozíciu $(x, y)$. A táto pozícia sa počas behu programu mení. Musíme teda meniť aj to, na akých súradniciach sa má vykresľovať konkrétny objekt? Našťastie nie. Jediné čo potrebujeme je naimplementovať triede metódu `nakresli(self,kreslic)`.

V rámci tejto metódy môžeme používať kreslic presne rovnakým spôsobom ako pred tým, jediné čo sa zmení je pozícia, kam sa budú dané obrázky kresliť. Kreslic v objekte, ktorý má pozíciu $(x, y)$ pokladá túto pozíciu za pozíciu $(0, 0)$. Zavolanie funkcie `kreslic.ciara((0,0),(5,5),2)` nenakreslí šikmú čiaru z rohu obrazovky, ale od pozície $(x, y)$. Môžete si to tiež predstaviť, že vždy keď chce kreslic niečo nakresliť, priráta si k daným súradniciam ešte súradnice $(x, y)$ a až potom to vykreslí na plochu.

### Príkaz `if`

Niekedy sa potrebujeme rozhodovať o veciach v závislosti na platnosti určitých podmienok. Napríklad, ak nám jedlo vypadne z obrazovky, teda jeho $y$-ová súradnica je väčšia ako výška okna, chceli by sme ju znova nastaviť na 0.

Na to nám bude slúžiť príkaz `if`.

```
if podmienka :
    prikaz1
    prikaz2
```

Syntax tohoto príkazu je jednoduchá. `Ak platí podmienka, tak sa vykoná postupne prikaz1 a prikaz2`.

Niekedy však chceme zareagovať aj na to, čo sa má stať ak nastane iná podmienka, poprípade chceme ošetriť všetky zvyšné možnosti. Na to sa použije `elif` a `else`. Nasledujúci program rozhodne o čísle *cislo*, či je kladné, záporné alebo nula.

```
if cislo > 0:
    print "kladne"
elif cislo < 0:
    print "zaporne"
else:
    print "nula"
```

Všimnite si, že za `else` nenasleduje podmienka, všetko čo nesedelo na predošlé podmienky je v tejto časti splnené. Takisto vykonávanie takéhoto zloženého `if`u je také, že postupne (od hora nadol) sa kontrolujú podmienky a vykonajú sa príkazy na **prvej** splnenej podmienke.

Ak by sme chceli vyjadriť komplikovanejšie podmienky, napr. číslo má byť kladné a párne, môžeme tieto dve podmienky spojiť pomocou slovíčka `and` (a).

`if cislo>0 and cislo%2==0:`

Táto podmienka je pravdivá iba ak sú pravdivé **obe** jej časti. Spojka `or` (alebo) je pravdivá ak je pravdivá **aspoň** jedna jej časť.

### Reagovanie na klávesnicu

Dôležitá časť každej hry je komunikácia s hráčom, teda reagovanie na stlečenie kláves. Na tento účel si `svet` pamätá pole `stlacene[]`. V tomto poli je pre každý kláves uložená hodnota `True` alebo `False`, podľa toho, či je daný kláves stlačený.

No konkrétnu klávesu sa opýtame tak, že do hranatých zátvoriek vložíme jej pomenovanie v špeciálnom formáte `pygame.K_nazovKlavesy`. Za *nazovKlavesy* môže ísť jednoduchý znak ako `'a'` alebo komplikovanejší zápis ako `DOWN` (pre šipku dole), `SPACE` pre medzerník ... V prípade, že potrebujete iné špeciálne znaky, najlepšie bude googliť.

```
if svet.stlacene[pygame.K_ESCAPE]:
    hra.koniec()
if svet.stlacene[pygame.K_w]:
    chodHore()
```

### Zrážanie

Bolo by veľmi vhodné, ak by sme vedeli detekovať zrážky a vyhodnocovať udalosti podľa nich. Napríklad ak sa ryba zrazí s jedlom, chce ho zožrať. Na naše šťastie, o toto všetko sa stará Gaminator takmer automaticky.

Majme triedu `Ryba` a triedu `Jedlo`. Ak chceme, aby sa ryba najedla, keď sa zrazí s jedlom, musíme napísať takýto program.

```
class Ryba(Vec):
    @priZrazke(Jedlo)
    def najemSa(self,jedlo):
        self.sytost += jedlo.vyzivnost
```

Riadok `@priZrazke(Jedlo)` hovorí, že nasledujúca funkcia ($najemSa()$) sa má zavolať ak sa ja (Ryba) zrazím s nejakým objektom z triedy Jedlo. No a funkcia $najemSa()$, ktorá sa volá po tomto zrazení, musí mať dva parametre – $self$, čo je premenná na tú konkrétnu rybu (pretože som v triede Ryba) a premennú *jedlo*, ktorá ukazuje na konkrétne jedlo, s ktorým sa tá ryba zrazila.

Posledná vec je, že ak chcem riešiť zrážky, potrebujem každému objektu určiť, akou časťou sa má zrážať. V našom prípade mu môžeme nastaviť len kolízny obdĺžnik a to pomocou štyroch premenných v $nastav()$ – $self.miestoHore$, $self.miestoDole$, $self.miestoVlavo$ a $self.miestoVpravo$. Tie hovoria, ako ďaleko od bodu $(x, y)$ daného objektu sa tento objekt ešte zráža. To vytvára obdĺžnik okolo bodu $(x, y)$.

O zrážanie týchto obdĺžnikov a volanie funkcií pri `@priZrazke(Trieda)` sa ďalej stará Gaminator.