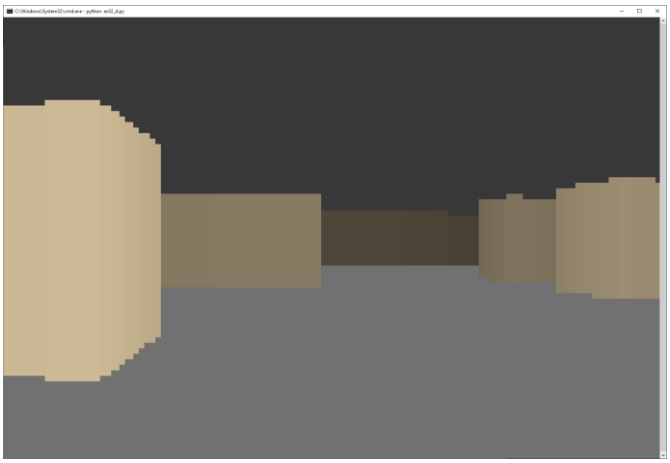
Cílem druhé úlohy je vytvoření základu jednoduché hry fungující na principu FPS [1, 2] v prostředí jazyka Python a příkazové řádky. Očekávaný výsledek by pro uvedenou mapu prostředí mohl vypadat takto:



```
"################",
"#              #",
"#              #",
"#              #",
"#######  ######",
"#              #",
"#       #######",
"#              #",
"#   ##         #",
"#              #",
"#              #",
"#      ##      #",
"#      ##      #",
"#  #########   #",
"#              #",
"################"
```

Pseudo 3D scéna je „vykreslena“ do textové konzoly pomocí ASCII znaků z pohledu hráče pohybujícího se v mapě reprezentované seznamem řetězců, kde znak `'#'` představuje zeď (kolize hráče se zdí není nutné řešit) a znak `' '` je volné políčko. Z algoritmického hlediska se jedná o nalezení bodu kolize paprsku vyslaného z aktuální pozice hráče a ve všech směrech jeho zorného pole se zvolenou překážkou, viz obrázek vpravo nahoře. Následující body by vás měly postupně provést detaily implementace tohoto úkolu.

Úloha 0: (0 bodů)

Vytvořte virtuální prostředí jazyka Python:

```
python -m venv tutorial-env
```

a aktivujte ho pomocí

```
tutorial-env\Scripts\activate          (win)
```

```
source ./tutorial-env/bin/activate     (linux)
```

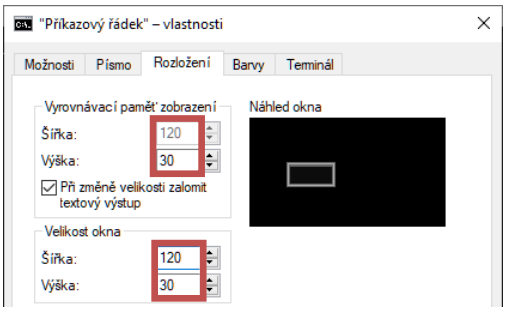
S využitím příkazu `pip` nainstalujte modul (knihovnu) pro monitorování stavu klávesnice a myši

```
pip install pynput
```

a modul pro ovládání barev a formátování výstupu do terminálu

```
pip install colored
```

Nastavte velikost okna příkazového řádku společně s velikostí vyrovnávací paměti zobrazení na požadovaný rozměr, např. 120 × 30 znaků. Použijete-li větší počet znaků a zmenšíte-li velikost písma, získáte větší rozlišení „obrazu“, ale program poběží pomaleji.



Přepište globální proměnné `width` (dále také jako w) a `height` (dále také jako h) ve výchozím skriptu na aktuální hodnoty velikosti vašeho okna. Při nesouladu těchto hodnot se skutečným počtem řádků a sloupců konzolového okna dojde k narušení výsledného „obrázku“.

Úloha 1: (1 bod)

Vytvořte seznam pojmenovaný `screen`, který bude reprezentovat obsah celého okna příkazové řádky po jednotlivých řádcích. Každý řádek bude reprezentován dalším seznamem, který bude tvořen znaky mezera (tj. `' '` nebo také `'\x20'`). Celkový počet prvků v seznamu `screen` bude odpovídat rozlišení `width` × `height` a jeho výsledná struktura bude následující:

```
[[' ', … , ' '], … [ … ]]
```

Tuto strukturu následně použijeme pro uložení jednotlivých „pixelů“ před jejich tiskem do konzoly.

Úloha 2: (1 bod)

Vytvořte funkci `move_cursor(r=1, c=1)`, která přesune kurzor na zvolený řádek a sloupec v konzoli. Přesun kurzoru z jeho aktuální pozice např. na požadované souřadnice provedete pomocí tzv. ANSI escape kódů či sekvencí [3] vytisknutím řetězce `'\x1b[r;cH'` do konzole příkazem `print`:

```
print('\x1b[{0};{1}H'.format(r, c), end='')
```

Obdobným způsobem dále vytvořte s pomocí příslušných řetězců funkce pro skrytí a zobrazení kurzoru a vykreslení barevného „pixelu“ s rozsahem celočíselných hodnot jednotlivých kanálu 0 až 255. Funkčnost naimplementovaných funkcí ověřte.

```
CURSOR_HIDE = u'\x1b[?25l'
CURSOR_SHOW = u'\x1b[?25h'
PIXEL = u'\x1b[48;2;{0};{1};{2};38;2;{3};{4};{5}m{6}\x1b[0m'
```

Pozn.: Před zadáváním ANSI escape kódů je nutné v konzoli zapnout jejich zpracování. Více viz ukázka ve výchozím skriptu na začátku funkce `main`.

Úloha 3: (1 bod)

Výpočet směru pohledu hráče.

Nejprve vypočítáme ohniskovou vzdálenost kamery f podle vzorce

$$f = \frac{w}{2\tan\left(\frac{fov_x}{2}\right)},$$

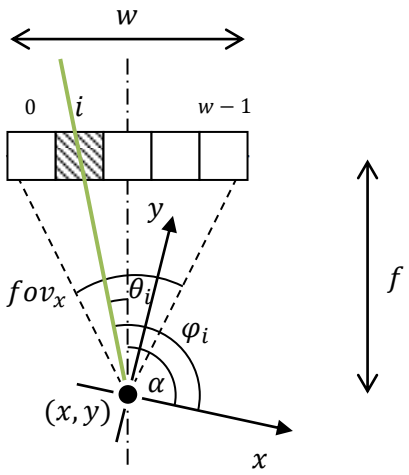
kde w je šířka obrazu, tj. počet sloupců konzoly (např. 120 znaků) a parametr fov_x je šířka zorného pole hráče v radiánech (např. 60°). Dále potřebujeme určit pomocný úhel θ_i, který svírá optická osa hráče a (zelený) parsek procházející středem i-tého sloupce konzoly podle vzorce

$$\theta_i = -\mathrm{atan}\left(\frac{i-w/2+0{,}5}{f}\right).$$

K tomuto úhlu připočteme aktuální natočení optické osy hráče α (směr pohledu hráče) a získáme úhel φ_i, který představuje absolutní natočení zeleného paprsku vůči souřadnému systému xy naší scény a můžeme tedy psát

$$\varphi_i = \alpha + \theta_i\,.$$

Celá situace je detailně zakreslena na následujícím obrázku. Zeleně zakreslenou polopřímku (paprsek) použijeme v následujícím kroku k určení viditelnosti nejbližší stěny skrze každý sloupec konzoly z pozice hráče.



Úloha 4: (1 bod)

Určení vzdálenosti stěny od pozice hráče pomocí tzv. ray marching algoritmu.

Algoritmus určení vzdálenosti nejbližší stěny od aktuální pozice hráče (x, y) dívajícího se směrem α je následující:

Pro každý i-tý sloupec konzoly:

a. Určíme úhel φ_i paprsku procházejícího daným sloupcem i.
b. S krokem 0,1 postupně generujeme ekvidistantní body (x_i, y_i) ležící na tomto paprsku. Je-li ve scéně na celočíselné pozici $(\lfloor x_i + 0{,}5\rfloor, \lfloor y_i + 0{,}5\rfloor)$ nalezen znak '#' indikující přítomnost stěny, potvrdíme existenci stěny v bodě (x_i, y_i) a zároveň

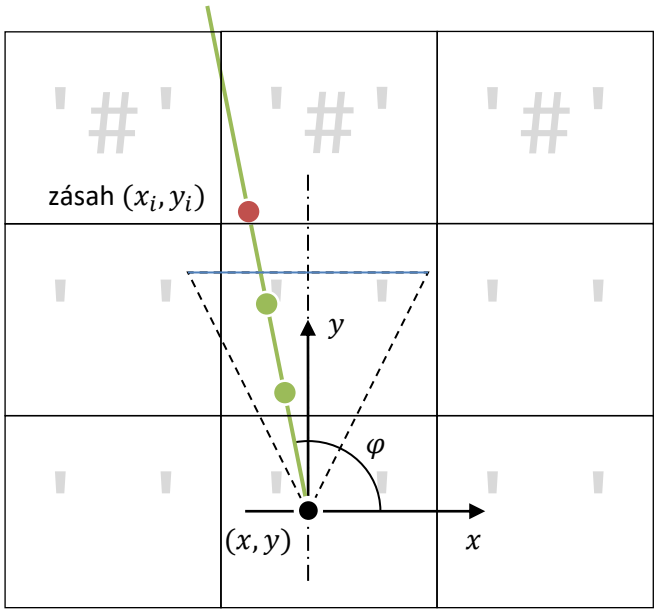
vrátíme aktuální hodnotu její vzdálenosti r_i od hráče. V opačném případě pokračujeme dále, dokud nepřekročíme meze scény.

Ekvidistantní body na zeleném paprsku můžeme generovat pomocí parametrické rovnice kružnice

$$\begin{aligned} x_i &= x + r_i \cos(\varphi_i) \\ y_i &= y + r_i \sin(\varphi_i) \end{aligned},$$

kde r_i je reálná hodnota z intervalu 0,1 až 23 s krokem 0,1.

Celý algoritmus ray marching by měl být také patrný z následujícího obrázku zachycujícího zmenšenou hrací mapu o rozměru 3 × 3 políčka, kde v horním řádku je umístěna stěna. Zelené body leží na místě, kde se ve scéně nachází znak mezera. Červený bod, do které jsme dospěli po třech krocích délky 0,1, tj. ve vzdálenosti 0,3 od aktuální pozice hráče ve směru i-tého sloupce konzoly, označuje místo nalezení nejbližší stěny ve scéně a algoritmus ray marching zde končí.



Pozn.: Číslo 23 představuje zaokrouhlenou největší možnou vzdálenost hráče od stěny pro čtvercovou plochu 16 × 16 políček a operace $\lfloor x + 0{,}5 \rfloor$ značí matematické zaokrouhlení.

Úloha 5: (1 bod)

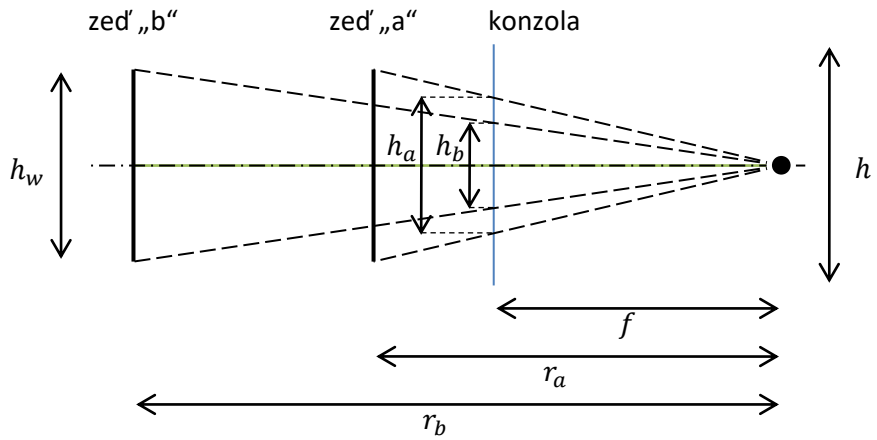
Vertikální vykreslení stěny do konzole.

Na základě určení vzdálenosti stěny r_i pro i-tý sloupec konzoly z předchozího kroku, můžeme vypočítat její zobrazenou výšku h_i z podobnosti trojúhelníků následovně

$$h_i = \frac{h_w f}{r_i},$$

kde h_w je skutečná výška stěny v jednotkách scény (např. 0,5) a f je již dříve vypočtená ohnisková vzdálenost oka hráče. Celá situace je zachycena na následujícím obrázku, kde jsou vyobrazeny dvě

stejně vysoké zdi v různých vzdálenostech, přičemž bližší zeď „a“ se bude jevit z pohledu hráče vyšší než zeď „b“ (je-li $r_a < r_b$ pak $h_a > h_b$; princip perspektivy).



Známe-li promítnutou výšku stěny h_i, můžeme do i-tého sloupce a příslušného řádku proměnné `screen` zapsat „pixel“ pomocí funkce vytvořené v prvním kroku. Sami zkuste promyslet způsob, jakým určíte, zda na daném řádku uvažovaného sloupce umístíte „pixel“ odpovídající stropu, zdi nebo podlaze. Pro zlepšení dojmu z hloubky 3D scény, můžete modulovat jas „pixelu“ v závislosti na vzdálenosti r_i bodu stěny, který reprezentuje.

Úloha 6: (1 bod)

„Vykreslení“ scény.

Závěrečným krokem je spojení všech řetězců z proměnné `screen` a jejich vypsání do konzoly. Před tímto krokem je nutné přesunout kurzor do levého horního roku pomocí funkce vytvořené v prvním kroku. Celá operace se opakuje v cyklu, dokud uživatel nezmáčkne např. klávesu `q` (jako Quit). Zároveň je v cyklu provedena úprava polohy a natočení uživatele podle aktuálního stavu proměnné `player`_action zavoláním vámi doplněné funkce `make`_action. Pro výpočet nové pozice hráče můžete opět využít funkci pro určení bodu na posunuté kružnici.

Reference

[1] https://en.wikipedia.org/wiki/First-person_shooter
[2] https://en.wikipedia.org/wiki/Wolfenstein_3D
[3] https://en.wikipedia.org/wiki/ANSI_escape_code