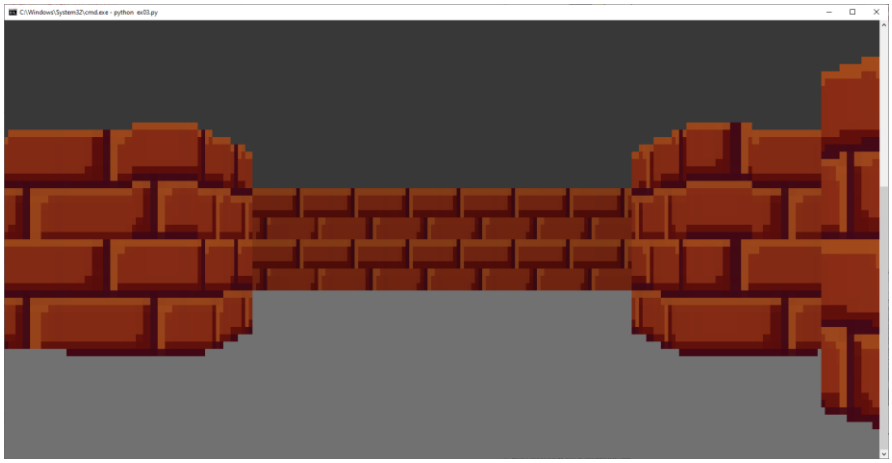
Cílem čtvrté úlohy je zkombinování výsledků druhého a třetího zadání a vytvoření otexturované verze jednoduché FPS hry. Očekávaný výsledek by pro texturu cihlové zdi mohl vypadat takto:



Textura bude načtena ze souboru uloženého v jednoduchém obrazovém formátu PPM [1, 2] a bude reprezentována instancí třídy Texture z modulu texture. Následující body by vás měly postupně provést detaily implementace tohoto úkolu.

Úloha 1: (1 bod)

Vytvoření třídy FPSEngine.

Navrhněte vhodnou strukturu nové třídy `FPSEngine`, která bude obsahovat všechny původně globální proměnné jako své instanční atributy. Dále tuto třídu rozšiřte o všechny potřebné metody. Nezapomeňte přesunout funkce `on_press` a `on_release` do nově vytvořené třídy, potřebujete v nich mít přístup k instančnímu atributu `player_action`. Funkce, které nijak nepracují s instančními atributy třídy `FPSEngine`, do této třídy nezahrnujte a ponechte je globální. Týká se to např. funkcí pro formátování ANSI escape kódů a parametrické kružnice. Očekávána podoba inicializace a spuštění hry bude následující:

```
def main():
  engine = FPSEngine(140, 40)
  engine.MainLoop()
```

Úloha 2: (1 bod)

Reprezentace herní plochy externím textovým souborem.

Do samostatného textového souboru ručně uložte obsah stávající (globální) proměnné `scene`. Třídu `FPSEngine` rozšiřte o metodu `LoadScene`, která načte zadaný textový soubor do instančního atributu `scene`. Formát reprezentace dat v této proměnné zůstane zachován. Nezapomeňte nastavit výchozí pozici hráče podle právě načtené mapy, např. do jejího středu. Očekávána podoba inicializace a spuštění hry pak bude následující:

```
def main():
  engine = FPSEngine(140, 40)
  engine.LoadScene('level_1.txt')
  engine.MainLoop()
```

Úloha 3: (1 bod)

Reprezentace akce hráče výčtovým typem [3].

Vytvořte třídu `PlayerAction` jakožto potomka třídy `Enum`. Názvy jednotlivých členů výčtového typu volte podle jejich sémantiky (např. `MOVE_FORWARD`) a zachovejte jejich původní hodnoty (např. `MOVE_FORWARD = 'w'`). Pomocí dekorátoru `@unique` zajistěte, že jednotlivé prvky výčtového typu budou vždy unikátní. Proveďte všechny další úpravy kódu, které jsou potřebné ke správnému využití právě vytvořeného výčtového typu.

Úloha 4: (1 bod)

Načtení textur z PPM souborů.

Naimportujte třídu `Texture` z module `texture`. Vytvořte metodu třídy `FPSEngine` pojmenovanou `LoadTexture`, která načte textury ze zadaných PPM souborů. Metoda bude přijímat libovolný počet dvojic (znak, název souboru). Využijte koncept předávání parametrů pomocí „*args“, tedy tzv. unpacking operátor [4]. Metoda uloží načtené textury do slovníku `textures`, kde klíčem bude uvedený znak symbolizující umístění dané textury ve scéně. Textový soubor se scénou upravte tak, aby v ní byly využity alespoň dvě různé textury. Očekávána podoba inicializace a spuštění hry pak bude následující:

```
def main():
  engine = FPSEngine(140, 40)
  engine.LoadScene('level_1.txt')
  engine.LoadTextures(('#', 'wall.ppm'), ('@', 'gold.ppm'))
  engine.MainLoop()
```

Úloha 5: (1 bod)

Nanesení textur na zdi.

Abychom mohli využít připravenou metodu `get_texel`, která přijímá texturovací souřadnice ve formě dvou reálných parametrů u a v z rozsahu $\langle 0,1 \rangle$, musíme nejprve tyto souřadnice vypočítat. Texturovací souřadnice vypočítáme na základě polohy zasažené zdi. Místo zásahu zdi pro každý paprsek jsme v algoritmu ray casting uložili do (x_i, y_i) [5]. Nyní si stačí dohodnout, s jakou frekvencí budou textury na zdi naneseny; řekněme, že jedna textura se na každém políčku scény zopakuje dvakrát na výšku i na délku. Finální výpočet texturovacích souřadnic pro daný bod na zdi by pak mohl vypadat následovně:

```
u = 2 * ((hit_x % 1 + hit_y % 1) % 1)
v = 2 * (j - tmp) / h_i
```

kde `hit_x` a `hit_y` je bod zásahu zdi (x_i, y_i), j je právě vykreslovaný řádek daného sloupce konzoly, `tmp` je pomocná proměnná určující výšku stropu, resp. podlahy a je vypočtena jako

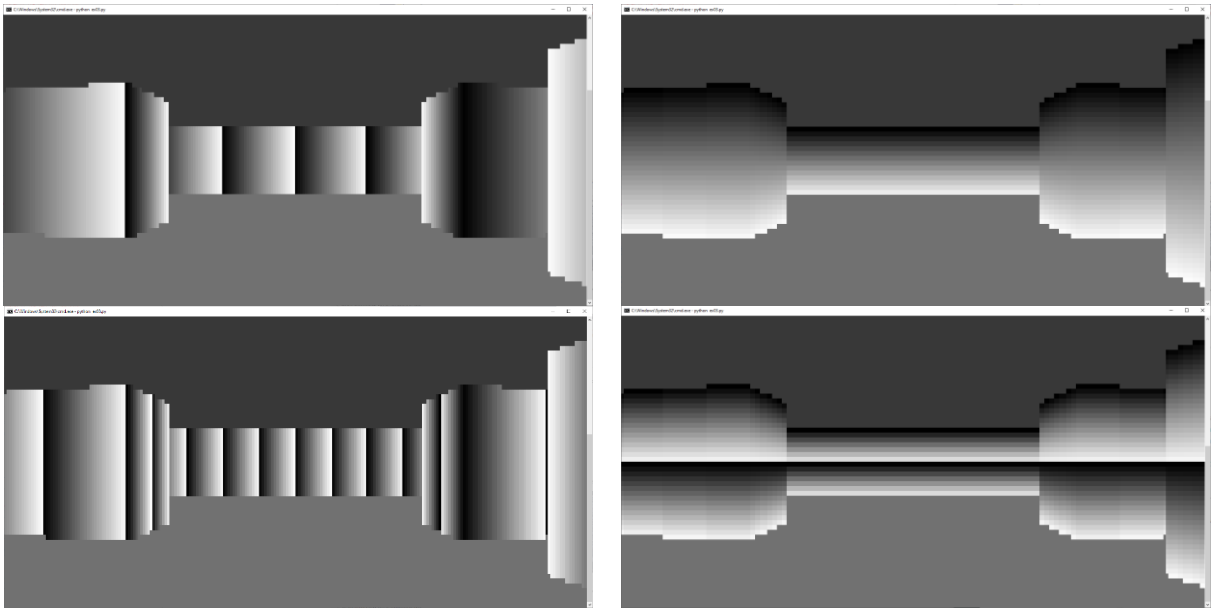
```
tmp = (height – h_i) / 2
```

kde `height` je počet řádku konzoly a h_i je vypočtená výška promítnuté stěny [5].

Samotné nanesení texelu vybrané textury do obrazového bufferu `screen` by pak mohlo vypadat následovně:

```
texel = self.__textures[self.__scene[int(hit_y)][int(hit_x)]].get_texel(u, v)

# ještě se potřebujeme dostat z rozsahu <0, 1> do <0, 255>; uvedená hodnota value
je rovna 255 nebo je zde uložena hodnota vhodným způsobem respektující vzdálenost
od zasažené stěny pro vytvoření dojmu mlhy

texel = (int(texel[0] * value), int(texel[1] * value), int(texel[2] * value))

# správně bychom měli zajistit, že hodnoty uložené v texelu budou zaokrouhleny
matematicky na celé číslo a vždy budou v rozsahu <0,255>; rozmyslete si vhodný
způsob, jak toho docílit

screen[j][i] = make_pixel(texel) # nakonec texel zapíšeme na odpovídající pozici
```

Pro vaši konkrétnější představu, jak texturování probíhá, si prohlédněte následující obrázky. První řádek zachycuje situaci, kdy se textura nanáší na každé políčko jedenkrát, druhý řádek odpovídá zopakování textury v obou směrech dvakrát. První sloupec odpovídá horizontální texturovací souřadnici u a druhý sloupec koresponduje s vertikální texturovací souřadnicí v.



Reference

[1] http://netpbm.sourceforge.net/doc/ppm.html
[2] https://en.wikipedia.org/wiki/Netpbm
[3] https://docs.python.org/3/library/enum.html
[4] https://towardsdatascience.com/unpacking-operators-in-python-306ae44cd480
[5] http://mrl.cs.vsb.cz/people/fabian/skj/ex02.pdf