Raycasting labyrint: Programátorská dokumentace

Filip Kastl

26. února 2021

Co to je

Jednoduchá first-person hra určená hlavně k tomu, abych předvedl svoji implementaci omezeného raycasting 3d enginu. Cílem hry je projít labyrint a najít vlajku – zlatou pásku, která reprezentuje konec úrovně.

Hráč má možnost pohybovat se dopředu, dozadu a do stran, ale nikoliv nahoru a dolů. Je možné rozhlížet se doleva a doprava, ale nahorů a dolů ne. Díky těmto limitacím nemusí v programu existovat výšková souřadnice. Vše se odehrává pouze na osách x a y, navzdory zdání program pracuje ve 2d.

Zdi jsou vykreslené jako lichoběžníky bez textury. Hráč by neměl mít možnost projít zdí. Layout labyrintu program při spuštění načítá z textového souboru.

Pro spuštění je potřeba:

- Python 3 (program vyvíjen v 3.6.9)
- Pygame 2 (program vyvíjen v 2.0.0)
- Numpy (program vyvíjen v 1.13.3)

Pojmy a souřadnicové konvence

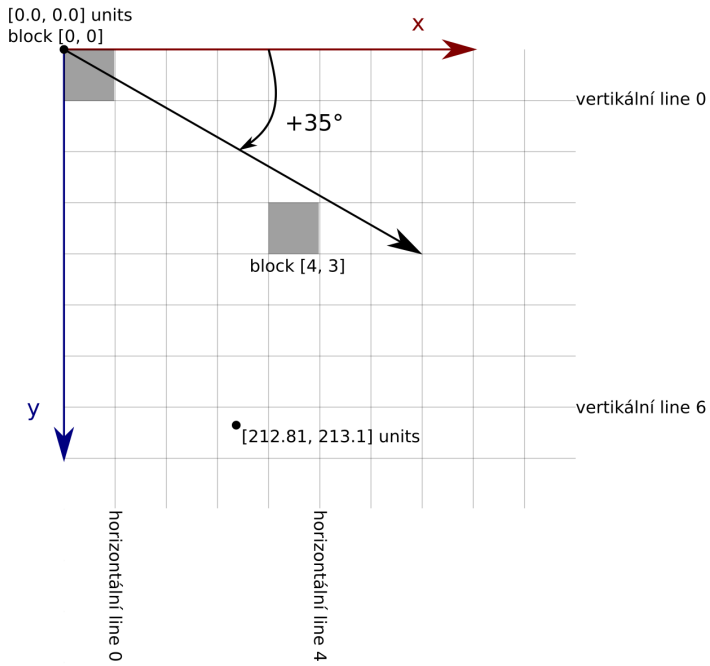
V dokumentaci budu mluvit o **paprscích** (rays ve slově raycasting). Paprsky si lze představit jako polopřímky vedoucí z hráčovy pozice. Paprsků je v $\_\_main\_\_.py$ definovaný fixní počet P. Směry paprsků rovnoměrně vyplňují 360 stupňů. Paprsky jsou identifikovány přirozenými čísly 0 až $P - 1$. Paprsek 0 je považovaný za následníka paprsku $P - 1$. Pokud některá funkce bere „ray“ jako argument, bere právě toto číslo paprsku.

Program používá dvojí jednotky souřadnic. Občas musí přepočítat jedny na druhé. **„Unit“ souřadnice** jsou „jemnější“ souřadnice. Po těchto souřadnicích se pohybuje hráč. **„Block“ souřadnice** jsou souřadnicemi ve kterých jsou zadávány a ukládány pozice zdí. $1\ block\ =\ 64\ units$.

V dokumentaci je také možné setkat se s vertikálními a horizontálními **„lines“**. Pokud si rozdělíme labyrint podle „block“ jednotek, „lines“ budou přímky na takto vzniklých předělech.

V názvech proměnných a komentářích zmiňuji **cataclysm**. To je můj název pro vizuální efekt, který nastane, když hráč dokončí labyrint. Během něho se pohyb hráče zpomaluje a postupně více a více paprsků začíná záměrně vracet špatné hodnoty vzdáleností, popř. nevracet žádné hodnoty. V dokumentaci se mu nebudu hlouběji věnovat.

Ilustrace souřadnicových konvencí



Osa x vede zleva doprava, osa y seshora dolů. Úhly jdou po směru hodinových ručiček.

Jak to funguje

Entrypointem programu je skript *__main__.py*. Zde jsou definované parametry, se kterými má být hra spuštěna (velikost okna, úroveň detailu, field of view). Skript vytvoří objekt labyrintu a objekt hry a spustí hlavní loop.

Třídy

Většina programu existuje v těchto následujících čtyřech třídách. Od každé třídy během svého běhu program vytváří pouze jeden objekt (uznávám, že je to možná trochu zvláštní design).

O načítání labyrintů se stará třída **Level**. Objekt této třídy při vzniku přečte soubor s labyrintem a z něho naparsovaná data v sobě uloží. Je pak schopný dát odpověď na dotazy „Je na těchto souřadnicích zeď/vlajka?“. Tato třída využívá výhradně „block“ souřadnice.

Game je hlavní třídou programu. Objekt této třídy během vzniku inicializuje raycasting logiku, objekt hráče, knihovnu Pygame (vytvoří okno, plátno, ...) a UI prvky (vykreslování textu, minimapa, ...). Nachází se v něm také hlavní loop hry, kde se mimo jiné odehrává vykreslování herního světa (viz metodu *Game.run()*).

Raycasting je největší třídou programu. Obsahuje veškeré informace o paprscích a všechny metody, jejichž účelem je pracovat s paprsky. Více v sekci *Algoritmy a výpočty*.

Nakonec je tu třída **Player**. Objekt této třídy uchovává informace o pozici hráče a orientaci kamery. Zároveň obsahuje funkce, které pohybují hráčem (a kontrolují, jestli nedošlo ke kolizi) nebo otáčejí kamerou. Tato třída využívá výhradně „unit“ souřadnice.

Významné metody

Level konstruktor

Kontruktor dostane cestu k souboru s labyrintem. Ten otevře a přečte z jeho prvního řádku šířku a výšku labyrintu v „block“ jednotkách. Vytvoří boolean dvojrozměrné pole s těmito rozměry. Projde zbytek řádků souboru a zaznačí si do pole na odpovídající místa, že se na nich má nacházet zeď. Uloží si také, na které pozici se nachází hráč a na které vlajka.

Game.run()

Tato metoda je hlavním loopem hry. Mimo jiné, sleduje čas, který uběhl od prvního pohybu hráče po moment, kdy prošel vlajkou. Každou iteraci provede následující:

- Pokud jsou stisknuté nějaké pohybové klávesy, pohne s hráčem zavoláním adekvátních metod třídy *Player*.
- Pokud je „block“ pozice hráče shodná s pozicí vlajky, ukončí hru.
- Pomocí *Raycasting.cast_rays()* vyšle všechny paprsky náležící do FOV hráče. Pokud paprsek zasáhl zeď a/nebo vlajku, vykreslí „column“ – úsek zdi (a/nebo vlajky) na obrazovku. Výška a odstín tohoto úseku závisí na vzdálenosti, kterou paprsek urazil, než něco zasáhl. **Tomuto způsobu vykreslování 3d se říká raycasting. Zde je implementovaný.**
- Vykreslí minimapu, FPS counter, výherní obrazovku (prošel-li už hráč vlajkou) a časovač.
- Přičte čas od posledního ticku do časovače a počká na další tick.

Raycasting konstruktor

Konstruktor dostane počet paprsků, se kterým má hra pracovat. Pro každý paprsek spočítá jeho úhel a jednotkový vektor (aby paprsky vyplnily 360 stupňů) a také jeho vertikální a horizontální „přepony“ (viz sekce *Algoritmy a výpočty*). Tyto hodnoty tedy není třeba počítat za běhu hlavního loopu hry.

Raycasting.cast_rays()

Nejdelší a nejdůležitější funkce programu. Ze zadané pozice (typicky pozice hráče) vystřelí paprsky v zadaném rozmezí (rozmezí je typicky FOV hráče). Každý paprsek postupně „prodlužuje“. Jakmile paprsek protne zeď, metoda si zapíše souřadnice průsečíku a vzdálenost, kterou paprsek urazil. Pokud paprsek mezitím protl i vlajku, zapíše si metoda i vzdálenost k vlajce.

Pokud paprsek vyletěl mimo labyrint nebo pokud urazil 10 „blocks“ aniž by protl zeď (popř. vlajku), metoda si místo vzdálenosti a průsečíku zapíše *None*.

Metoda ve skutečnosti vystřelí každý paprsek dvakrát. Jednou kontroluje průsečíky s vertikálními stranami zdí (protíná paprsek a vertikální „lines“) a jednou průsečíky s horizontálními (protíná paprsek a horizontální „lines“). K čemu je to dobré bude zodpovězeno v sekci *Algoritmy a výpočty*.

Metoda vrací tři seznamy. Seznam vzdáleností k průsečíku se zdí, seznam samotných průsečíků se zdí a seznam vzdáleností k průsečíku se zdí.

Raycasting.fisheye_coefficients()

Raycasting vykreslování produkuje efekt „rybího oka“ – zeď se nevykreslí rovně, ale směrem k okraji okna se scvrkává. Tento efekt lze vyrušit, pokud se pro každý paprsek (podle toho, jaké pozici uvnitř FOV odpovídá) vzdálenost, kterou pro něj vrací *cast_rays()*, vynásobí správným koeficientem. Koeficienty během spuštění hry generuje tato funkce.

Raycasting ostatní metody

Raycasting třída také obsahuje metody, které odpovídají na dotazy typu „Který paprsek je o 4 paprsky nalevo od tohoto?“ nebo „Který paprsek je opačný tomuto?“ a také metody, které pracují s úhly a jednotkovými vektory paprsků.

Player pohybové metody

Má-li se hráč pohnout dopředu, dozadu nebo do stran, je zavolána odpovídající metoda. Objekt třídy *Player* si pamatuje, který paprsek momentálně míří „dopředu“. Podle něho pak může nechat třídu *Raycasting* dopočítat i ostatní směry.

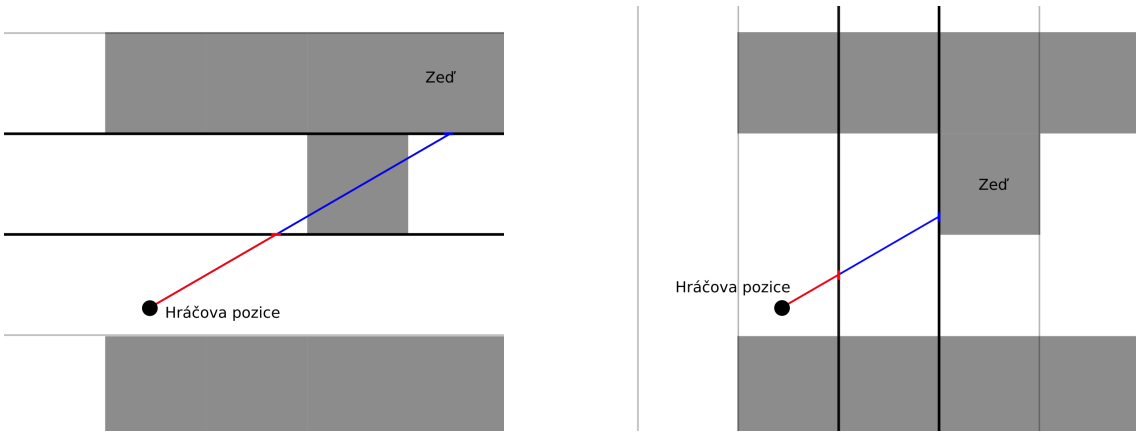
Než se hráč pohne, je zavolána metoda *Raycasting.cast_rays()* na jediný paprsek. Pokud vzdálenost, kterou paprsek urazí, je menší, než vzdálenost, o kterou by se hráč měl pohnout, hráč se nepohne. Takto program implementuje, že zdi nemají být prostupné.

Player.turn()

Kamera hráče začíná orientována ve směru osy x. Orientaci třída *Player* zaznamenává jako číslo paprsku, který míří „kupředu“. Zároveň si pamatuje i který paprsek se nachází na levém a pravém okraji FOV. Tato metoda otočí kameru o nějaký počet paprsků n – tedy přičte toto n ke všem třem zmíněným paprskům.

Algoritmy a výpočty

Raycasting aneb „vystřelení paprsku“



Jak již bylo zmíněno, vystřelení paprsku probíhá vždy dvakrát. Jednou program kontroluje průsečíky s horizontálními stranami zdí a jednou s vertikálními. To nám umožňuje použít trik, který snižuje náročnost celé operace vystřelení. Mimochodem – jak je vidět na prvním obrázku – toto také vede k neintuitivním situacím, kdy paprsek v jedné (zde v horizontální) fázi raycastingu zcela mine nejbližší zeď a teprve až druhá fáze tuto „chybu“ opraví.

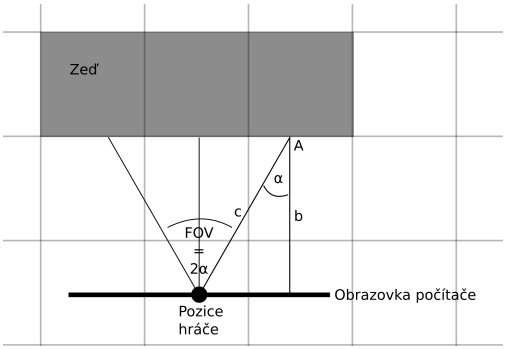
Program nemusí kontrolovat každý „unit“, kterým paprsek prochází, ale jen každý „block“ (zeď nikdy menší než „block“ nebude). To znamená, že vlastně dělá průsečíky paprsku a „lines“. Nám však stačí udělat vždy jen jeden průsečík, a to s tou nejbližší „line“ od hráče ve směru paprsku (znázorněno červeně). Poté zkontrolujeme, jestli se bezprostředně za touto „line“ nenachází zeď. Pokud ano, máme hotovo. Pokud ne, paprsek prodlužujeme.

Program však má už předpočítáno o jaký konkrétní vektor je třeba paprsek prodloužit, aby dosáhl na další vertikální/horizontální „line“ (znázorněno modře), jelikož už při startu programu známe úhel daného

paprsku a vzdálenost mezi „lines“. Stačí tedy tento vektor (v kódu těmto vektorům říkám „hypotenuses“) přičíst a znovu zkontrolovat, jestli jsme nenarazili na zeď. Tento proces program opakuje, dokud nedojde ke zdi nebo nevyčerpá limit pro to, kolikrát lze paprsek natáhnout (standardně 10 prodloužení).

Program takto získá „horizotální“ průsečík a „vertikální“ průsečík. Z nich vybere ten, který je blíže k hráči.

Rybí oko



Paprsky hra vystřeluje z bodu. Jenomže obrazovka počítače není bod, ale rovná plocha. Chceme-li například zjistit vzdálenost k bodu A (na obrázku), získáme vystřelením paprsku vzdálenost c. Hra využívá vztahu $b = cos(\alpha) \cdot c$, aby vzdálenost opravila (α je polovina FOV, které má hra běžně nastavené na 60 stupňů).

Bugy, o kterých vím

- Občas se nevykreslí roh zdi a lze vidět skrz.
- Hráč bohužel má možnost projít zdí. Pokud se napozicuje u rohu zdi tak, aby se roh nevykreslil, může v tom místě projít skrz.
- Pohyb šikmo je rychlejší než pohyb rovně.

Tyto bugy jsem se rozhodl neřešit. Dle mého dodávají alespoň trochu hloubky jinak docela nudné hře. Důkazem nechť je to, že jsem strávil (neúspěšně) dost času pokusy o překonání času 15s v *2.lvl* používáním právě těchto bugů.

Co si dál přečíst

Zde je tutoriál, který vysvětluje princip raycastingu a jak ho implementovat. Neřídil jsem se jím zcela, ale rozhodně mě navedl na správnou cestu. Kupříkladu trik s předpočítanými vektory mám právě z tohoto tutoriálu. Je dle mého čtivý a dostatečně podrobný.