Gi

Vyšší odborná škola a Střední průmyslová škola elektrotechnická Olomouc, Božetěchova 3

**PRAKTICKÁ ZKOUŠKA Z ODBORNÝCH PŘEDMĚTŮ**

2D Hra v Pythonu

Autor **Matouš Zatloukal**

Obor **Elektrotechnika**

Vedoucí práce **Mgr. Jaroslav Krbec**

Školní rok **2021/2022**

Na druhé stránce bude vloženo originální zadání.

*(včetně podpisů vedoucího práce a maturujícího žáka)*

Prohlašuji, že jsem praktickou zkoušku z odborných předmětů vypracoval samostatně a všechny prameny jsem uvedl v seznamu použité literatury.

……………………………  
 Matouš Zatloukal

Chtěl bych vyslovit poděkování panu Mgr. Jaroslavu Krbci za odborné konzultace a poskytnuté informace.

……………………………  
 Matouš Zatloukal

Prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé práce nebo její části se souhlasem školy.

……………………………  
 Matouš Zatloukal

Abstrakt

Tato práce je zaměřena na kompletní vývoj počítačové hry včetně veškeré logiky a animací. Videohry jsou stále populárnější formou zábavy a to nejen u mladších generací. Zároveň roste i velikost herního průmyslu, kde je pozice programátora jednou z těch nejdůležitějších a nejžádanějších.

Cílem práce je vytvořit 2D hru, která bude obsahovat základní herní mechaniky, jako například pohyb, skákání nebo útočení. Vymyslet logiku těchto mechanik a naprogramovat je ve zvoleném jazyce. Dále pak vytvořit animace postav, za které budou uživatelé hrát, a nakonec naprogramovat interaktivní menu a propojit vše dohromady do jednoho programu.

Toho lze docílit použitím některé z grafických knihoven programovacího jazyku Python, kde zejména dobrá je knihovna Pygame, jelikož ta je specificky navržená pro vývoj her. Dále je pak využit program Aseprite, který je ideálním nástrojem na tvoření jednoduché pixelové grafiky a následné animování.

Hra je, stejně jako velké množství her dnešní doby, určena pro zábavu uživatelů. Slouží ale také pro mne, jakožto autora, abych si vyzkoušel programování bez použití jakýchkoliv dalších nástrojů, které jsou v herním průmyslu běžně využívány a které práci výrazně ulehčují. Je to hlavně z důvodu pochopení základních herních mechanik a nabytí znalostí o jejich funkčnosti.

Obsah

[Abstrakt 4](#_Toc97572076)

[Obsah 5](#_Toc97572077)

[Úvod 7](#_Toc97572078)

[1. Užitý software 8](#_Toc97572079)

[1.1 Python 8](#_Toc97572080)

[1.2 Pygame 8](#_Toc97572081)

[1.3 OS 9](#_Toc97572082)

[1.4 Random 9](#_Toc97572083)

[1.5 Aseprite 10](#_Toc97572084)

[2. Logika hry a herní mechaniky 11](#_Toc97572085)

[2.1 Herní logika 11](#_Toc97572086)

[2.1.1 Pohyb 11](#_Toc97572087)

[2.1.2 Útoky 12](#_Toc97572088)

[2.1.3 Animace 15](#_Toc97572089)

[2.1.4 Skákání 17](#_Toc97572090)

[2.1.5 Kolize 17](#_Toc97572091)

[2.1.6 Health bar 19](#_Toc97572092)

[2.2 Menu 19](#_Toc97572093)

[2.2.1 Parametr *game* 20](#_Toc97572094)

[2.2.2 Smyčka *MainMenu* 20](#_Toc97572095)

[2.2.3 Kontrola událostí v menu 22](#_Toc97572096)

[2.2.4 Smyčka po konci hry 23](#_Toc97572097)

[3. Animace 25](#_Toc97572098)

[3.1 Inspirace a vytvoření prvního snímku 25](#_Toc97572099)

[3.2 Vytváření animací 27](#_Toc97572100)

[3.3 Implementace animací do hry 29](#_Toc97572101)

[Závěr 31](#_Toc97572102)

[Seznam použité literatury 32](#_Toc97572103)

[Seznam obrázků a tabulek 33](#_Toc97572104)

[Přílohy 35](#_Toc97572105)

Úvod

V této práci se budu zaměřovat na vytvoření 2D hry pomocí objektově orientovaného programování v jazyce Python a jeho modulu Pygame bez pomoci jakýchkoliv herních enginů. Budu zároveň tvořit i interaktivní menu, přes které se bude hra spouštět a do kterého se uživatelé budou vždy vracet.

Ve hře budu užívat i své vlastní obrázky kreslené stylem „pixel art“, které budu animovat v editoru obrázků Aseprite.

Hra samotná se bude odehrávat v aréně, kde mezi sebou budou bojovat dva hráči. Každý hráč bude představovat jednu z těchto postav. Ty se od sebe budou lišit jejich dosahem a rychlostí útoků. Cílem každého z hráčů bude snižovat počet životů protivníka, dokud nedosáhne nuly.

Rozhodl jsem se pro tuto práci hlavně z důvodu mého budoucího studia vývoje počítačových her a zlepšení svých znalostí v tomto oboru. Ty si můžeme zahrát, když si zrovna potřebujeme odpočinout, nebo si jen tak užít volný čas s přáteli či rodinou.

# Užitý software

Před tím, než se pustím do popisování vytváření animací, herní logiky nebo samotného programování, bude nutné nejprve uvést mnou užité softwarové nástroje pro vývoj hry.

## Python

Python je vysokoúrovňový, interpretovaný programovací jazyk navržený v roce 1991. Je vyvíjen jako open source projekt, který nabízí instalační balíky pro většinu běžných platforem (Unix, MS Windows, macOS, Android).



Obrázek 1 - Logo programovacího jazyku Python

Zdroj: https://www.python.org/community/logos/

Tento programovací jazyk jsem si vybral hlavně z důvodu, že k němu mám nejblíže, a tedy ho i nejvíce ovládám. Vhodnější by ovšem bylo použít například C++ nebo C#, které jsou k vytváření videoher mnohem lépe uzpůsobené.

## Pygame

Pygame je modul programovacího jazyku Python, který slouží k vytváření a designování videoher. Stejně jako Python je i Pygame podporován na většině hlavních platforem.



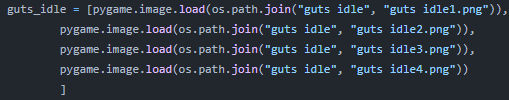
Obrázek 2 - Logo modulu Pygame

Zdroj: https://github.com/pygame/pygame/blob/main/docs/pygame\_logo.png

Programování pomocí modulu Pygame se dělí na dvě části. Deklarování proměnných, příprava funkcí, popř. tříd a poté na herní smyčku, která se neustále opakuje a využívá naše deklarované proměnné, funkce nebo i třídy. V této smyčce také probíhá samotná hra.

## OS

Pomocí modulu OS lze provádět různé interakce s uživatelským operačním systémem. V mé práci jsem tohoto modulu využíval velmi zřídka, a to jen tehdy, když jsem potřeboval načíst obrázky uložené v různých složkách.



Obrázek 3 - Využití modulu OS

Zdroj: Vlastní zpracování

## Random

Modul Random je jeden z nejrozšířenějších Python modulů vůbec. Slouží ke generování pseudonáhodných čísel, což jsou čísla, jejichž posloupnost se zdá být náhodná, ale ve skutečnosti je generována deterministickým algoritmem.

Tento modul jsem podobně jako modul OS využíval jen velmi zřídka, a to pouze, když se náhodně vybírá pozadí arény, ve které se bude hrát.

## Aseprite

Aseprite je program, za jehož pomoci je možné vytvářet obrázky a animace ve stylu pixel art. Stejně jako většina programů na úpravu obrázků podporuje Aseprite vrstvení. Celý program je vyvíjený za pomoci programovacího jazyka C++.



Obrázek 4 - Logo programu Aseprite

Zdroj: <https://github.com/aseprite/aseprite/blob/main/data/icons/ase256.png>

Pro zjednodušení animací pak také umožňuje ukládat jednotlivé obrázky do snímků a zobrazovat jich více najednou.

Exportování je taktéž velmi snadné, jelikož program umí exportovat do GIF souborů stejně, jako třeba do PNG nebo JPG.

# Logika hry a herní mechaniky

V této kapitole si představíme logiku herních mechanik, jako jsou například pohyby, kolize nebo skákání. Ukážeme si zde také, jak funguje herní menu.

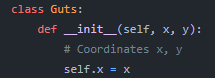
## Herní logika

Podkapitola Herní logika bude zaměřená na hru samotnou, hernímu menu se bude věnovat následující podkapitola.

### Pohyb

U většiny her je pohyb tou nejdůležitější mechanikou a tady tomu není jinak. Jedna z vlastností modulu Pygame je možnost získávat vstupy z kláves na klávesnici. Díky tomu budeme moci monitorovat některé námi specifikované klávesy a poté na jejich zmáčknutí reagovat různými akcemi postav ve hře.

Souřadnice X a Y si budeme ukládat do příslušných proměnných.

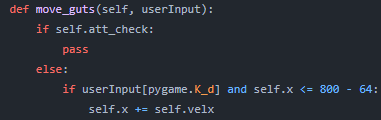


Obrázek - Ukládání souřadnic

Zdroj: Vlastní zpracování

Prozatím budeme potřebovat pouze souřadnici X, jelikož se budeme pohybovat pouze v ose X na kartézské soustavě souřadnic, se kterou Pygame pracuje. Souřadnici Y budeme potřebovat až v kapitole, která se bude zabývat skákáním.

V herní smyčce pak budeme každou iteraci aktualizovat námi později vytvořenou proměnnou *userInput*, která nám bude zjišťovat vstupy z klávesnice od obou hráčů. Pomocí hodnot této proměnné budeme měnit i pozice hráčů.



Obrázek 6 - Změna souřadnic postavy

Zdroj: Vlastní zpracování

Na tomto snímku je vidět, že pokud je do proměnné *userInput* zaznamenáno zmáčknutí klávesy *d* a postava není pryč z herního okna, které je široké 800 pixelů, tak se k souřadnici X přičte námi definovaná hodnota ve funkci *\_\_init\_\_*.

### Útoky

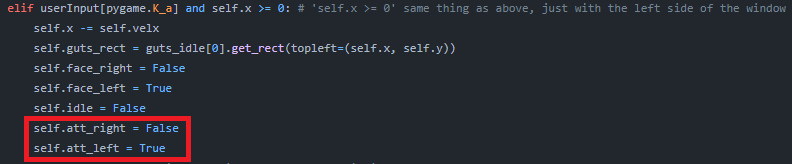
Útoky se budou dělit podle toho, s kterou postavou útočíme. Jedna z nich, Guts, totiž útočí pouze na vzdálenost dosahu svého meče, mezitím co druhá, Mrakoplaš, střílí ohnivé koule po celé šířce arény.

Při útočení s Gutsem budeme potřebovat následující proměnné, které nám pomohou zjistit, na kterou stranu má být při animaci útoku natočen a kdy započít útok.

Bez názvu.png

Obrázek - Proměnné kontrolující útoky

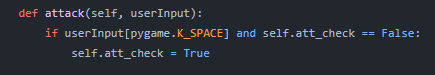
Tyto proměnné se nacházejí ve funkci *\_\_init\_\_* ve třídě *Guts. Att\_right* a *att\_left* se mění při běhání s postavou na pravou či levou stranu.



Obrázek 8 - Příklad změny proměnné při pohybu doleva

Zdroj: Vlastní zpracování

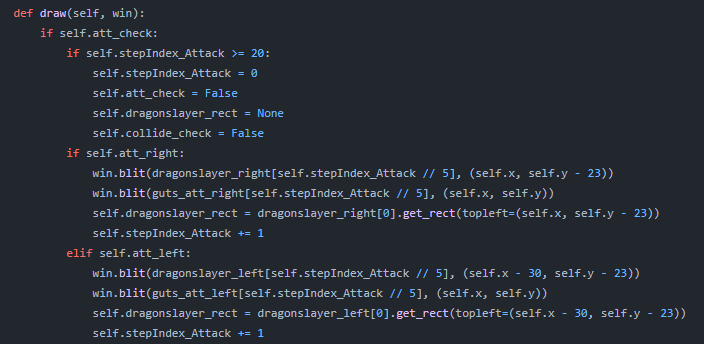
Pomocí proměnné *userInput* opět zjistíme, zda je naše požadovaná klávesa stlačena a pokud tomu tak je, nastavíme *att\_check* na True.



Obrázek 9 - Funkce útočení třídy Guts

Zdroj: Vlastní zpracování

Také se nesmí zapomenout ohlídat, aby postava nemohla útočit, pokud už útočí. Pokud tedy víme na kterou stranu se postava dívá a víme, zda hráč zmáčkl příslušnou klávesu, můžeme začít vykreslovat animaci útoku.

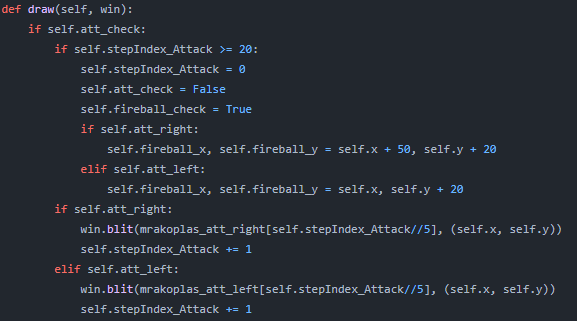


Obrázek 10 - Vykreslení útoku Gutse

Zdroj: Vlastní zpracování

Nejprve se provede kontrola, zda se už nedokončila animace. Tím by byl ukončen útok, jelikož toto vykreslování probíhá ve smyčce. Následně se zkontroluje, zda se postava dívá na levou či pravou stranu a poté se vykreslí animace těla a meče v daném snímku na dané souřadnice. U této postavy je animace meče a těla samotného rozdělena, jelikož později, až budeme chtít provádět kolize, budeme chtít kolidovat Mrakoplaše pouze s mečem a ne s celým tělem Gutse, což se bude dělat jednodušeji, pokud animace budou rozděleny na dvě.

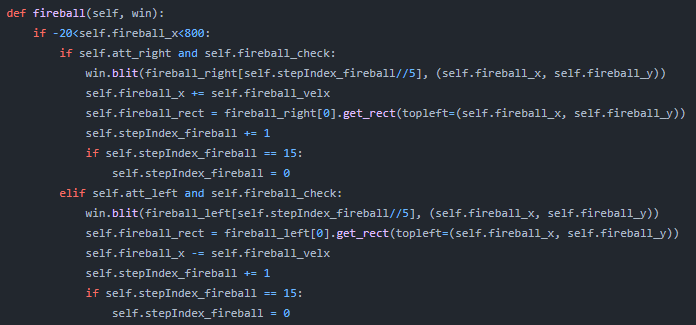
Krom proměnných *att\_right*, *att\_left* a *att\_check* budeme u Mrakoplaše potřebovat i proměnnou *fireball\_check*, která se opět nachází ve funkci *\_\_init\_\_*.



Obrázek 11 - Vykreslení útoku Mrakoplaše

Zdroj: Vlastní Zpracování

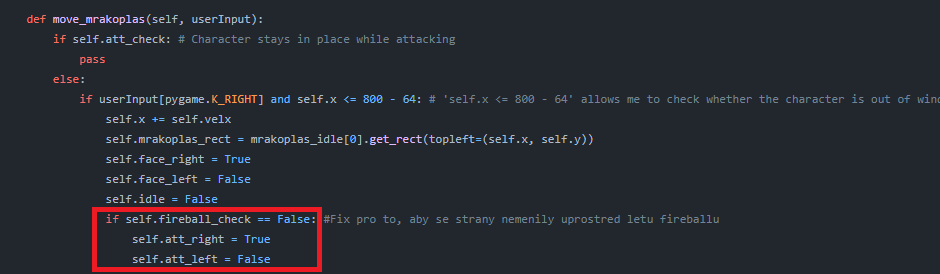
Po ukončení animace se aktivuje proměnná *fireball\_check* a přejde se k vykreslování samotné ohnivé koule, která bude poté kolidovat s tělem Gutse a působit poškození.



Obrázek 12 - Vykreslování projektilu Mrakoplaše

Zdroj: Vlastní zpracování

Při změně proměnných *att\_right* a *att\_left* se ovšem musí ošetřit, aby hráč nemohl měnit směr projektilu po jeho vystřelení.



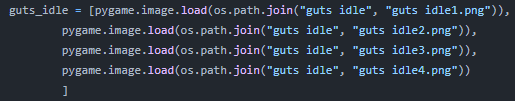
Obrázek 13 - Ošetření změny směru projektilu

Zdroj: Vlastní zpracování

Proměnné se nám tedy mění pouze, pokud projektil není vystřelený. Projektil mizí, pokud koliduje s druhou postavou, nebo se dostane za hranu arény.

### Animace

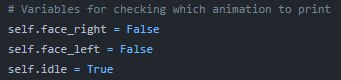
Obrázky jednotlivých animací si prvně musíme nahrát.



Obrázek 14 - Nahrání obrázků pro animace

Zdroj: Vlastní zpracování

Ve funkci *\_\_init\_\_* nalezneme proměnné *face\_right*, *face\_left* a *idle*, které nám pomohou sledovat, které animace pohybů se právě mají přehrávat.



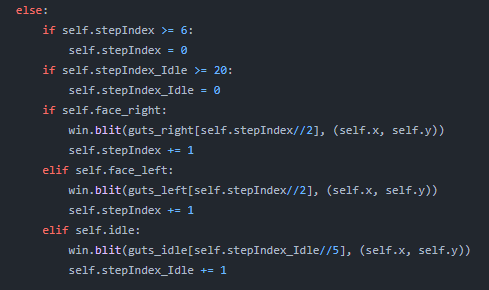
Obrázek 15 - Proměnné pro animace pohybu

Zdroj: Vlastní zpracování

Tyto proměnné se mění na základě toho, zda naší postavou pohybujeme a popřípadě kam.

Další proměnnou, kterou budeme potřebovat je *stepIndex*. Ten nám pomůže zjistit, který snímek animace se má právě vykreslit, jelikož pomocí této proměnné budeme poukazovat do listu obrázků na snímek, který požadujeme.

Ve funkci *draw* pak uplatníme všechny námi připravené proměnné.



Obrázek 16 - Funkce draw

Zdroj: Vlastní zpracování

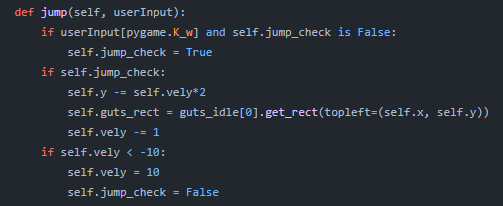
Vykreslování opět probíhá ve smyčce. Nejprve zkontrolujeme, zda se *stepIndex* nedostal k námi stanovené maximální hodnotě, kde další hodnota už by ukazovala mimo list a nastala by tedy chyba. Pokud ano, tak ho vynulujeme. Poté zjistíme, zda se postava pohybuje doleva či doprava a vykreslíme jeden snímek animace. Nakonec přičteme ke *stepIndexu* jedničku, aby při příští iteraci poukazoval na následující obrázek v listu, a tím vytvoříme animaci.

Může se ovšem stát, že animace budou vypadat příliš rychlé. To je zapříčiněno malým počtem snímků v animaci, rychlým zobrazováním snímků na obrazovku a nebo kombinací těchto dvou věcí. V mém případě zobrazuji na obrazovku 60 snímků za sekundu a moje animace má maximálně 5 snímků.

Musím tedy vykreslovat stejný snímek během více iterací, aby animace nevypadala tak nepřirozeně a rychle. Toho docílíme tak, že vynásobíme počet snímků určitým přirozeným číslem, tím číslem následně budeme dělit *stepIndex*, který poukazuje na snímek při vykreslování, a výsledek tohoto násobení stanovíme jako maximální hodnotu *stepIndexu* při které se nuluje. Jeden obrázek nám tedy zůstane vykreslen déle a animace si takto můžeme libovolně protahovat.

### Skákání

Jednou z dalších důležitých mechanik je skákání. To nám umožní se lépe pohybovat po aréně.



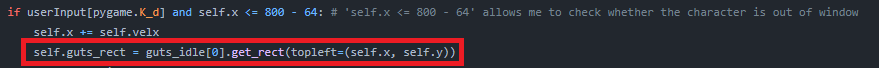
Obrázek 17 - Funkce jump

Zdroj: Vlastní zpracování

Pomocí proměnné *userInput* je opět načítán klávesový vstup. Také se před začátkem skoku musí zkontrolovat, zda už postava neskáče. Pokud je načten správný klávesový vstup a postava neskáče, započne odčítání od proměnné *y*, která zaznamenává pozici na ose Y. Velikost proměnné, která je odčítána od proměnné *y*, se zmenšuje každou iteraci až do doby, kdy se čísla převrátí na záporné, a začne se tedy přičítat. Díky tomu docílíme efektu gravitace, kde čím výše je postava ve vzduchu, tím pomaleji stoupá, až se zastaví úplně, a začne klesat. Při docílení opačné hodnoty *vel.y* se postava zastaví a proměnná se vyresetuje. Tím docílíme toho, že postava skončí na stejné pozici Y, jako když začal skok.

### Kolize

Před vytvářením kolizí mezi postavami je nutné vytvořit rectangle objekty zbraní, ohnivých koulí a těl postav. Rectangle objekty slouží v Pygamu k ukládání a manipulaci s obdélníkovými oblastmi na obrazovce.

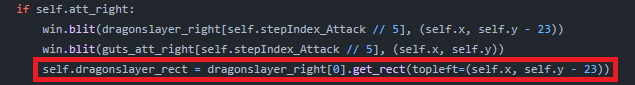


Obrázek 18 - Rectangle objekt postavy Guts

Zdroj: Vlastní zpracování

Zde je vidět aktualizování pozice rectanglu těla Gutse při jeho pohybu doprava. Jako obrázek je možné uvést jakýkoliv snímek jakékoliv animace, jelikož všechny mají velikost 64 pixelů. Souřadnice se ovšem musí uvést aktuální, aby se rectangle pohyboval zároveň s pohybem postavy.

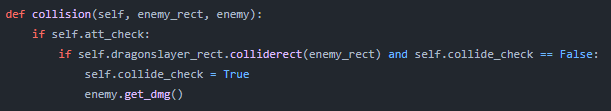
Při útoku Gutse se jako rectangle nastaví prostor celé animace jeho meče.



Obrázek 19 - Rectangle objekt Gutsova meče

Zdroj: Vlastní zpracování

Jako obrázek může být uveden opět kterýkoliv snímek animace, musí se ovšem dát pozor, aby to byla animace správného směru útočení. V tomto případě animace útoku doprava. Jako souřadnice jsou uvedeny upravené souřadnice příslušné postavy tak, aby animace útoku meče byla seřazená vůči animaci útoky postavy samotné.



Obrázek 20 - Funkce collision

Zdroj: Vlastní zpracování

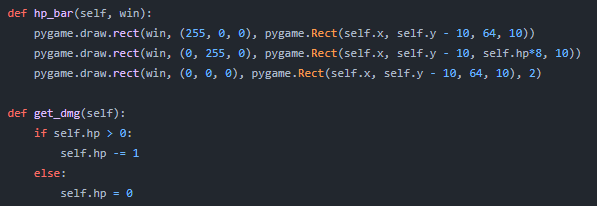
Nyní tyto dva rectangly mohou kolidovat. Do funkce jako *enemy\_rect* předáme rectangle nepřítele (tedy Mrakoplaše). A jako *enemy* předáme funkci objekt Mrakoplaše, aby se mohla vyvolat funkce *get\_dmg*.

Při kolizi Gutse a ohnivé koule od Mrakoplaše je postup stejný s tím rozdílem, že se rectangle koule vytvoří až po jejím seslání a ne už při samotné animaci.

### Health bar

Na kolize bude hra reagovat ubíráním určitého počtu životů z maxima dané postavy. Tuto reakci budeme zobrazovat pomocí ukazatele životů.

Začneme tedy vykreslením barevného ukazatele.



Obrázek 21 - Funkce hp\_bar a get\_dmg

Zdroj: Vlastní zpracování

Při každé iteraci herní smyčky budeme vykreslovat červený pásek, zobrazující chybějící životy postavy. Ten bude překrýván zeleným páskem, zobrazujícím počet životů, který postupným zmenšováním na základě vyhodnocení kolize bude odhalovat červený pásek, a tím indikovat ztracené životy. Nakonec vykreslíme černé ohraničení pro lepší viditelnost ukazatelů.

Funkci *get\_dmg* voláme pokud nastane kolize, aby odečetla určitý počet životů. Tyto dvě funkce nám společně zajistí, aby postavy ztrácely životy, když kolidují s nepřátelskou zbraní a aby se tyto životy správně vykreslovaly na obrazovku.

## Menu

V této podkapitole se budu věnovat principům fungování interaktivního menu.

Přes menu mohou uživatelé spustit nebo vypnout hru a taktéž se sem vracejí, pokud byl dohrán již dříve spuštěný zápas.

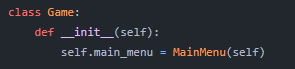
### Parametr *game*

Parametr *game* je jedním ze základních kamenů potřebných pro fungování mnoha funkcí v mém programu. Při definování funkcí třídy *Game*, která slouží, jako herní smyčka, chceme častokrát použít proměnné a funkce třídy *MainMenu*. Toho lze docílit pomocí vytvoření objektu *MainMenu*, na který pak můžeme jednoduše odkazovat v objektu *Game.* Chceme ovšem zároveň využívat proměnné a funkce třídy *Game* při definování funkcí ve třídě *MainMenu*. Pokud tedy přidáme třídě MainMenu parametr *game*, do kterého později v objektu *Game* vložíme sebe samého neboli *self*, můžeme používat všechny funkce a proměnné námi definované ve třídě *Game*.

Bez názvu.png

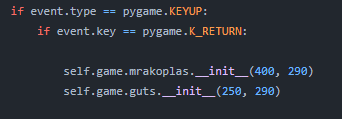
Obrázek 22 - Paramert game u třídy MainMenu

Zdroj: Vlastní zpracování



Obrázek 23 - Vytváření objektu MainMenu

Zdroj: Vlastní zpracování

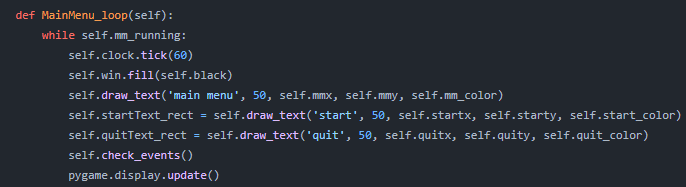


Obrázek 24 – Příklad odkazování pomocí game argumentu ve třídě MainMenu

Zdroj: Vlastní zpracování

### Smyčka *MainMenu*

Když se program spustí, musí se ihned vstoupit do nějaké smyčky, ve které se setrvá, dokud nebude zahájen zápas nebo ukončena hra, a do které se hráči budou vracet po dohrání zápasu. Právě takovou smyčkou se bude zabývat tato podkapitola.



Obrázek 25 - Funkce MainMenu smyčky

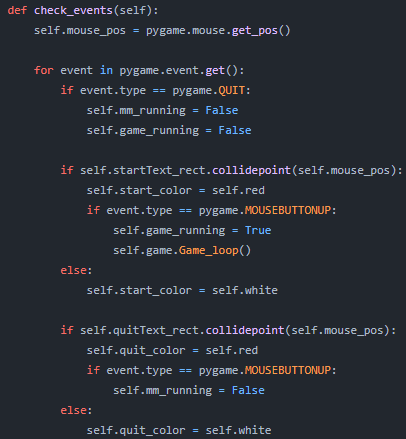
Zdroj: Vlastní zpracování

V této smyčce vykonává program následující akce neustále dokola:

* uzpůsobí rozestup mezi snímky tak, aby byl počet snímků za sekundu maximálně 60 (zespodu nelze hranici omezit, jelikož tam závisí na výkonnosti počítače)
* vyplní okno černou barvou
* nakreslí text „main menu“
* nakreslí text „start“ a zaznamená ho jako rectangle, se kterým se později bude moct kolidovat
* nakreslí text „quit“ a taktéž ho zaznamená jako rectangle
* provede funkci *check\_events*
* aktualizuje okno, aby se provedly všechny změny pro daný snímek

### Kontrola událostí v menu

Funkce *check\_events* slouží ke kontrole událostí, neboli „eventů“, jak se jim říká v Pygamu. Za tyto eventy se považuje jakákoliv akce provedená uživatelem, jako např. stisk klávesy nebo pohyb myší.



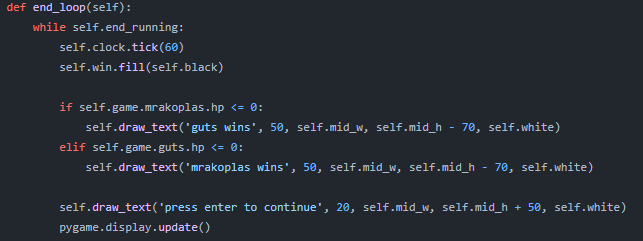
Obrázek 26 - Funkce check\_events

Zdroj: Vlastní zpracování

V naší funkci se nejprve zaznamenají souřadnice myši pro daný snímek, jelikož je budeme později potřebovat při kolidování. Poté funkce zkontroluje, zda uživatel náhodou nevypnul okno a pokud ano, ukončí všechny možné probíhající smyčky. Následně proběhne kontrola kolize s textem „start“. Pokud myš a rectangle textu kolidují, tak se text vybarví červeně. Pokud uživatel během kolize zmáčkne tlačítko na myši, spustí se herní smyčka. Kolize s textem „quit“ fungují stejně, akorát při zmáčknutí tlačítka na myši se hra vypne.

### Smyčka po konci hry

Poslední věc, která chybí bezproblémovému přechodu z menu do hry a zpět je *end\_loop*. Funkce *end\_loop* se nachází ve třídě *Menu* a slouží, jako smyčka, ve které program setrvá, jakmile jeden z hráčů v zápase klesne pod 0 životů.

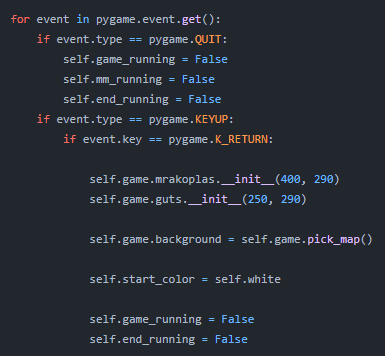


Obrázek 27 - První část funkce end\_loop

Zdroj: Vlastní zpracování

V této smyčce program vykonává následující akce:

* uzpůsobí rozestup mezi snímky tak, aby byl počet snímků za sekundu maximálně 60
* vyplní okno černou barvou
* zkontroluje, která z postav má počet životů 0 a podle toho vykreslí text oznamující výherce zápasu
* vykreslí text indikující uživateli, jak pokračovat zpět do menu
* aktualizuje obrazovku



Obrázek 28 - Druhá část funkce end\_loop

Zdroj: Vlastní zpracování

Ve druhé části funkce *end\_loop* probíhá kontrola eventů, které mohou nastat při iterování touto smyčkou. Prvně se kontroluje, zda uživatel nezavřel okno a případně se zastaví všechny spuštěné smyčky. Následně se kontroluje, jestli uživatel nestiskl klávesu *return* neboli *enter* a pokud ano, vytvoří se nové objekty postav, které nahradí ty stávající, to hlavně z důvodu resetování všech proměnných, které se postavě v zápase měnily, poté se vybere jedno ze dvou pozadí arén a vypnou se smyčky *end\_loop* a *Game\_loop*. Tím se program vrátí do smyčky *MainMenu\_loop*, odkud mohou uživatelé znovu zahájit zápas.

# Animace

Plynulejší pocit při hraní dodávají animace. V této kapitole se budeme zabývat animacemi postav při stání na místě, běhu nebo útoku, které využívám v mém programu. Zejména vytvářením jednotlivých snímků, jejich animováním a implementací do hry. V této kapitole se ovšem nebudu zabývat pozadím arén ve hře, jelikož ty nejsou tvořeny mnou z důvodu náročnosti kreslení.

## Inspirace a vytvoření prvního snímku

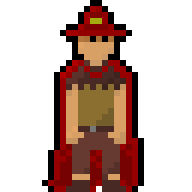
Při tvorbě mých postav jsem se do značné míry inspiroval charaktery z mých oblíbených knih. Jména jsem oběma postavám zanechal stejná, jako v knihách.



Obrázek 29 – Ilustrace Mrakoplaše od Paula Kidbyho

Zdroj: Kniha Výtvarné umění Zeměplochy

První z nich je Mrakoplaš z knižního cyklu Zeměplocha od autora Terryho Pratcheta, kterého jsem překreslil pomocí využití stylu pixel art. Obrázek jsem poté zvětšil do rozlišení 64 pixelů.



Obrázek 30 - Překreslení Mrakoplaše pomocí pixel art

Zdroj: Vlastní zpracování

Druhou z postav je Guts z manga série Berserk, kterou psal a kreslil Kentaró Miura.



Obrázek 31 - Panel z mangy Berserk

Zdroj: Manga Berserk Vol. 29

Toho jsem taktéž stylem pixel art překreslil do obrázku, který jsem později zvětšil na 64 pixelů.



Obrázek 32 - Překreslení Gutse pomocí pixel art

Zdroj: Vlastní zpracování

## Vytváření animací

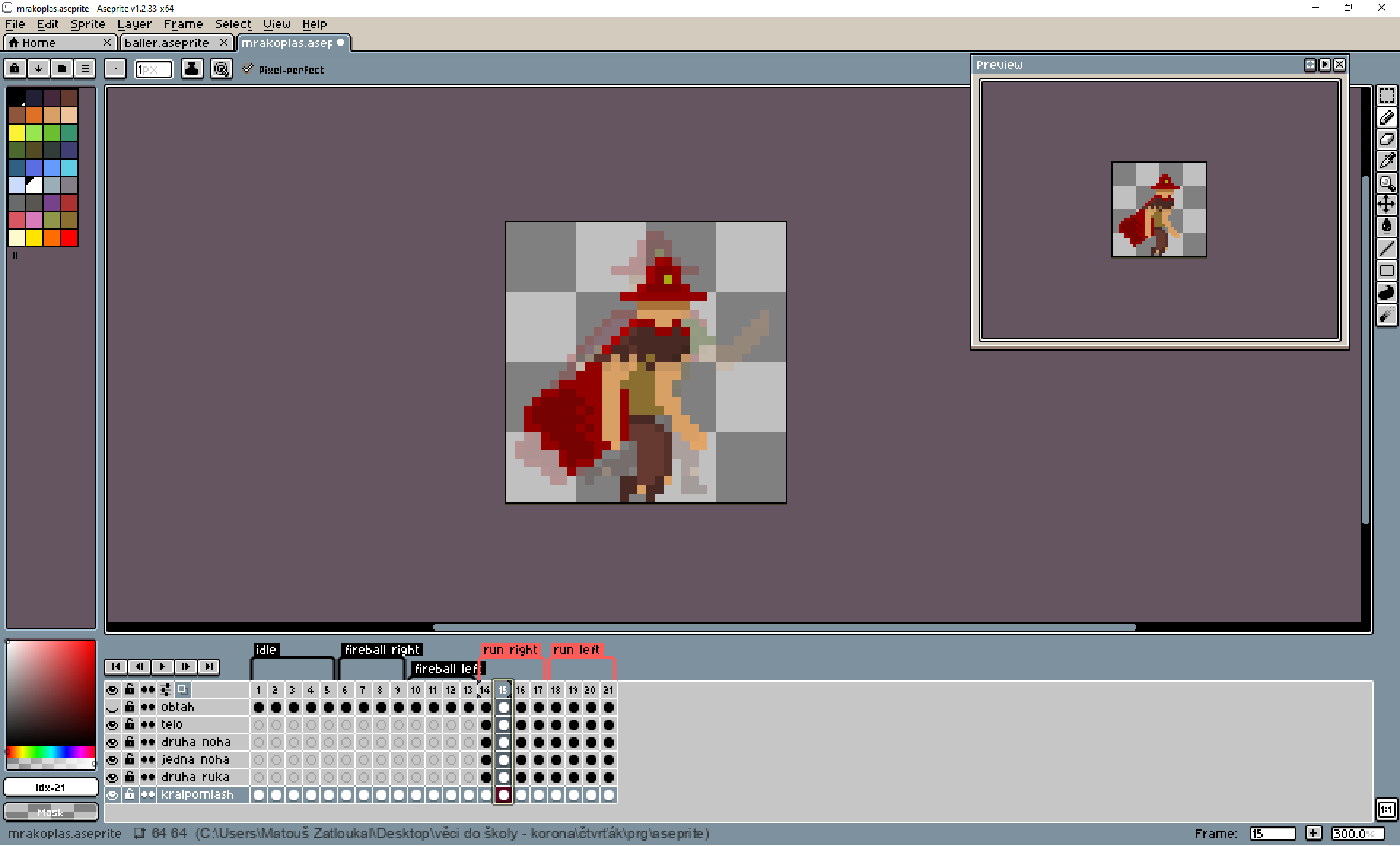
Jelikož je animace jen sled obrázků, vykreslených rychle za sebou, bude potřeba vytvořit další snímky.



Obrázek 33 - Snímky animací Mrakoplaše

Zdroj: Vlastní zpracování

Zde jsou vyobrazeny všechny snímky animací Mrakoplaše krom těch, které jsou pouze vertikálně převráceny (např. běh doleva).



Obrázek 34 - Kreslení animací

Zdroj: Vlastní zpracování

Při tvoření animací budeme využívat hlavně funkci zobrazování předchozích snímků která nám pomůže při vymýšlení pohybů pro následující snímek.



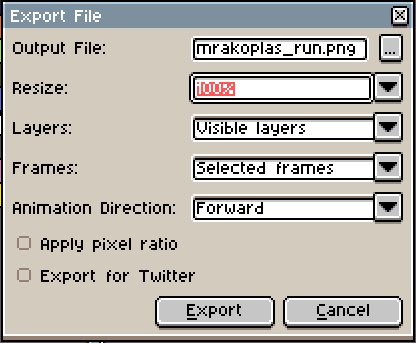
Obrázek 35 - Snímky animací Gutse

Zdroj: Vlastní zpracování

Zde můžeme lépe vidět, jak je rozdělena tělo Gutse při útoku a jeho meč. Kolidování Mrakoplaše s mečem je pak mnohem jednodušší na realizaci, jelikož pouze vezmeme rectangle meče.

## Implementace animací do hry

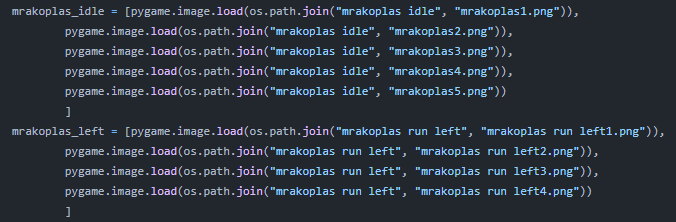
Animace jsou z programu Aseprite exportovány do obrázků ve formátu png, vždy se jeden snímek animace rovná jednomu exportovanému obrázku.



Obrázek 36 - Exportování v programu Aseprite

Zdroj: Vlastní zpracování

Takto exportované obrázky seřadíme do seznamů v našem programu, aby se daly následně používat při animacích a my na ně do seznamu mohli odkazovat pomocí *stepIndexů*.



Obrázek 37 - Načtení obrázků do listů v programu

Zdroj: Vlastní zpracování

Závěr

S celkovým výsledkem práce jsem velmi spokojen. Svou znalost knihovny Pygame a schopnost s ní pracovat už jsem si několikrát ověřil na menších projektech a v této práci jsem všechny tyto zkušenosti zužitkoval a ještě nabyl několik nových, jako třeba práce s animacemi nebo vytváření postav s různými dosahy útoků.

Při vytváření logických aspektů hry jsem postupoval velmi systematicky a nestalo se mi, že bych narazil na nějaké závažnější chyby. Vždy jsem začal s nápadem, jak by daná mechanika mohla fungoval. Pokud bylo potřeba, načrtnul jsem si na papír jednotlivé kroky, které se ve herní smyčce budou opakovat, abych si tuto myšlenku lépe představil. Následně jsem mechaniku vyzkoušel, a pokud fungovala, udělal jsem snímek na stránku GitHub, kam jsem celý svůj projekt ukládal. Takto jsem postupoval při vytváření veškeré herní logiky.

Animování vlastních postav byla jedna z úloh, které jsem se na první pohled bál, jelikož jsem neměl skoro žádné zkušenosti, jak s vytvářením grafických částí her, tak s jejich animováním. Díky výpomocným videím na platformě YouTube, které se zaměřují na práci s programem Aseprite se mi ale podařilo nakreslit jednoduché modely postav a vytvořit základní animace jejich pohybu. Pozadí různých arén nejsou kreslená mnou, jelikož jsem si netroufnul do něčeho tak velkého s tak malými zkušenostmi. Jsou tedy staženy pod licencí, která je dovoluje volně distribuovat.

Naprogramování interaktivního menu a jeho propojení se samotnou hrou bylo těžší, než jsem očekával. To převážně z důvodu mého rozložení funkcí menu a hry do dvou samostatných souborů a využíváním souboru menu jako knihovny pro soubor se hrou.

Nenarazil jsem ovšem na nic, co by vývoj pozastavilo na více, než jeden den. Do budoucna bych se rád soustředil na práci s herními enginy a momentálně se věnuji studiu jazyka C++. Také bych rád zlepšil své schopnosti při tvorbě grafické stránky hry, alespoň na úroveň, kdy si budu moct vytvořit veškeré potřebné materiály pro vývoj jednoduché hry sám, a to i včetně pozadí.

Seznam použité literatury

[1] *Pygame documentation* [online]. Dostupné z: <https://www.pygame.org/>

Seznam obrázků a tabulek

[Obrázek 1 - Logo programovacího jazyku Python 8](#_Toc97823220)

[Obrázek 2 - Logo modulu Pygame 9](#_Toc97823221)

[Obrázek 3 - Využití modulu OS 9](#_Toc97823222)

[Obrázek 4 - Logo programu Aseprite 10](#_Toc97823223)

[Obrázek 5 - Ukládání souřadnic 11](#_Toc97823224)

[Obrázek 6 - Změna souřadnic postavy 12](#_Toc97823225)

[Obrázek 7 - Proměnné kontrolující útoky 12](#_Toc97823226)

[Obrázek 8 - Příklad změny proměnné při pohybu doleva 12](#_Toc97823227)

[Obrázek 9 - Funkce útočení třídy Guts 13](#_Toc97823228)

[Obrázek 10 - Vykreslení útoku Gutse 13](#_Toc97823229)

[Obrázek 11 - Vykreslení útoku Mrakoplaše 14](#_Toc97823230)

[Obrázek 12 - Vykreslování projektilu Mrakoplaše 14](#_Toc97823231)

[Obrázek 13 - Ošetření změny směru projektilu 15](#_Toc97823232)

[Obrázek 14 - Nahrání obrázků pro animace 15](#_Toc97823233)

[Obrázek 15 - Proměnné pro animace pohybu 15](#_Toc97823234)

[Obrázek 16 - Funkce draw 16](#_Toc97823235)

[Obrázek 17 - Funkce jump 17](#_Toc97823236)

[Obrázek 18 - Rectangle objekt postavy Guts 18](#_Toc97823237)

[Obrázek 19 - Rectangle objekt Gutsova meče 18](#_Toc97823238)

[Obrázek 20 - Funkce collision 18](#_Toc97823239)

[Obrázek 21 - Funkce hp\_bar a get\_dmg 19](#_Toc97823240)

[Obrázek 22 - Paramert game u třídy MainMenu 20](#_Toc97823241)

[Obrázek 23 - Vytváření objektu MainMenu 20](#_Toc97823242)

[Obrázek 24 – Příklad odkazování pomocí game argumentu ve třídě MainMenu 20](#_Toc97823243)

[Obrázek 25 - Funkce MainMenu smyčky 21](#_Toc97823244)

[Obrázek 26 - Funkce check\_events 22](#_Toc97823245)

[Obrázek 27 - První část funkce end\_loop 23](#_Toc97823246)

[Obrázek 28 - Druhá část funkce end\_loop 24](#_Toc97823247)

[Obrázek 29 – Ilustrace Mrakoplaše od Paula Kidbyho 25](#_Toc97823248)

[Obrázek 30 - Překreslení Mrakoplaše pomocí pixel art 26](#_Toc97823249)

[Obrázek 31 - Panel z mangy Berserk 26](#_Toc97823250)

[Obrázek 32 - Překreslení Gutse pomocí pixel art 27](#_Toc97823251)

[Obrázek 33 - Snímky animací Mrakoplaše 27](#_Toc97823252)

[Obrázek 34 - Kreslení animací 28](#_Toc97823253)

[Obrázek 35 - Snímky animací Gutse 28](#_Toc97823254)

[Obrázek 36 - Exportování v programu Aseprite 29](#_Toc97823255)

[Obrázek 37 - Načtení obrázků do listů v programu 30](#_Toc97823256)

[Obrázek 38 - Poster 35](#_Toc97823257)

Přílohy



Obrázek 38 - Poster