STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Vývoj grafického uživatelského rozhraní v Pythonu

Jáchym Mierva
Kraj Vysočina

Žďár nad Sázavou 2020

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Vývoj grafického uživatelského rozhraní v Pythonu

Development of graphical user interface in Python

Autor: Jáchym Mierva
Škola: Biskupské gymnázium, Žďár nad Sázavou, U Klafárku 3
Kraj: Kraj Vysočina

Žďár nad Sázavou 2020

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

Ve Žďáře nad Sázavou dne 15. 3. 2020 ……

Jáchym Mierva

Anotace

Tato práce se zabývá programováním grafických aplikací pomocí vysokoúrovňové, objektově orientované knihovny pygame_widgets, kterou jsem vytvořil. Tato knihovna je rozšířením populární nízkoúrovňové knihovny pygame. Knihovna pygame_widgets je v této práci podrobně zdokumentována a vysvětlena. Kromě toho naznačuje práce další vývoj knihovny a možnosti jejího využití. Knihovna již byla použita pro vývoj hry Pampuch.

Klíčová slova

grafické uživatelské rozhraní; objektově orientované programování; dokumentace; vývoj her; Python

Annotation

This work deals with programming graphics applications using the high-level, object-oriented pygame_widgets library I created. This library is an extension of the popular low-level pygame library. The pygame_widgets library is documented and explained in detail in this work. In addition, the work suggests further development of the library and its use. The library has already been used in the development of Pampuch.

Keywords

graphical user interface; object-oriented programming; documentation; game development; Python

Obsah

1 Úvod

V této práci jsem se rozhodl shrnout poznatky, které jsem získal během programování knihovny pygame_widgets. Podstatnou částí této práce je také její vysvětlení a kompletní dokumentace.

V průběhu svého studia na osmiletém gymnáziu jsem se začal učit programovat v jazyce Python [1]. Poté, co jsem naprogramoval několik aplikací na bázi textové konzole, jsem se chtěl naučit programovat GUI. Moje první pokusy s grafikou se děly za pomoci knihovny Turtle, poté jsem se zkoušel naučit pracovat s knihovnou Tkinter. Příliš jsem tomu však nerozuměl, neměl jsem tehdy ani žádné zkušenosti s objektově orientovaným programováním. Následně jsem se setkal s knihovnou pygame, která mě velmi oslovila. Jak název napovídá, pygame je v mnoha ohledech ideální pro tvorbu her; programování nějaké komplexnější aplikace by však bylo nepřiměřeně náročné. Této knihovně jsem ve své práci věnoval celou kapitolu.

V září roku 2018 jsem se začal trochu učit programovat s pomocí PyQt5 [2]. Vzápětí mě však napadlo naprogramovat si vlastní rozšíření knihovny pygame. Při vytváření pygame_widgets jsem se mírně inspiroval tím, co jsem se stihl naučit o PyQt5, velmi často jsem však improvizoval. Nyní, po roce práce, má pygame_widgets více než 2000 řádků kódu. Rozhodl jsem se tedy napsat rozsáhlou dokumentaci. Ta by měla zásadním způsobem pomoct ostatním programátorům a potenciálním uživatelům mé knihovně porozumět a případně ji i editovat (viz kapitolu 6.4). Nezanedbatelným důvodem k sepsání dokumentace však bylo i to, že jsem si často ani já nepamatoval některé detaily potřebné k používání knihovny. Musel jsem v takové situaci prohledávat zdrojový kód a znovu se snažit porozumět implementaci různých funkcí. Snažil jsem se tedy v dokumentaci hlavně kvůli sobě zmínit i ty nejdrobnější detaily.

2 VYSOKOÚROVŇOVÉ A NÍZKOÚROVŇOVÉ PROGRAMOVACÍ JAZYKY

Přestože je procesor nejkomplexnější elektronický obvod, který byl kdy lidmi vyroben, podporuje pouze několik základních operací – základní aritmetiku, operace s pamětí a komunikaci s periferiemi. Funguje tedy úplně jinak než lidský mozek. Proto byly vytvořeny programovací jazyky, tj. jazyky, kterým rozumí člověk i počítač, a tedy nás můžou s počítači sblížit.

Programovací jazyky můžeme rozdělit na nízkoúrovňové (např. Assembler, C) a vysokoúrovňové (např. Java, C#, Python). První ze zmíněných skupin obsahuje jazyky, které jsou blíže k počítači než člověku. Nevýhodou těchto jazyků je, že mohou být pro programátora neintuitivní a hůře čitelné, i velmi jednoduchý program může mít mnoho řádků. Ukáži to na tzv. Hello World! programu, což je program, který na obrazovku vypíše text „Hello World!“. Takto vypadá v Pythonu:

```
1  print("Hello World!")
```

A takto v Assembleru (kód citován z [3]):

```
1  section     .text
2  global _start     ;must be declared for linker (ld)
3
4  _start:                ;tells linker entry point
5  mov   edx,len     ;message length
6  mov   ecx,msg     ;message to write
7  mov   ebx,1       ;file descriptor (stdout)
8  mov   eax,4       ;system call number (sys_write)
9  int   0x80        ;call kernel
10
11 mov   eax,1       ;system call number (sys_exit)
12 int   0x80        ;call kernel
13
14 section     .data
15 msg db 'Hello, world!', 0xa  ;string to be printed
16 len equ $ - msg     ;length of the string
```

Jak bylo vidět v předchozím příkladu, vysokoúrovňové jazyky jsou intuitivnější a obecně jednodušší na naučení. Je možné pomocí nich vytvořit v krátkém čase i relativně komplexní programy.

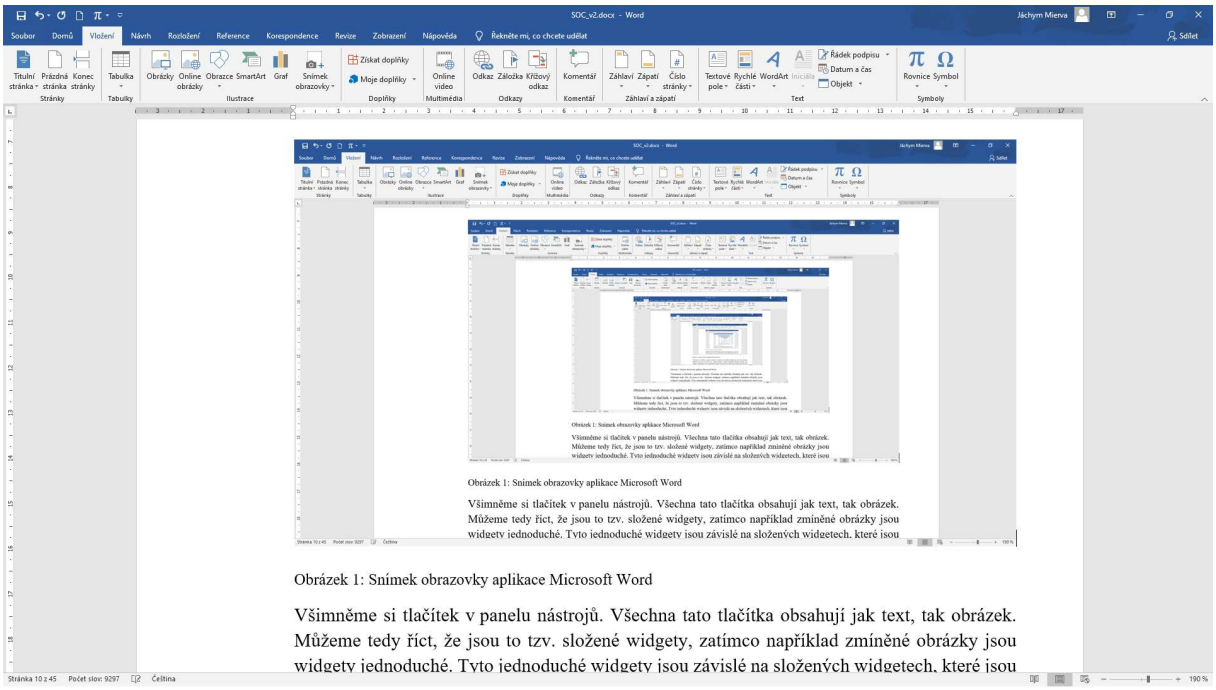
Jejich zásadní nevýhodou je však znatelně nižší rychlost, a to z důvodu neexistence optimálních kompilátorů. Kompilátor je program, který překládá (kompiluje) kód napsaný člověkem do strojového kódu, který je čitelný pro počítač. Nízkoúrovňové programovací jazyky jsou obecně velmi blízké strojovému kódu, avšak vysokoúrovňové jazyky používají několik vrstev abstrakce, která znemožňuje doslovný překlad. Kompilátor poté většinou nenalezne nejoptimálnější způsob, jak kód přeložit, což vyústí v situaci, kdy počítač vykonává velké

množství operací zbytečně. Neustálý vývoj kompilátorů se snaží kompilaci optimalizovat, stále však platí, že kód napsaný v Assembleru může být i 100x rychlejší než kód v Pythonu.

3 OBECNÁ STRUKTURA GUI

Základním pojmem v počítačové grafice je widget. Je to jakýkoli, většinou interaktivní, objekt zobrazený na obrazovce, např. tlačítko, obrázek, menu nebo i celé okno.

Určitě všichni víme, jak vypadá tlačítko nebo obrázek. Nyní bych však chtěl upozornit na jeden důležitý koncept. Widgety v okně jsou uspořádány ve stromovité struktuře, která velmi usnadňuje práci programátora. Pojďme si to ukázat na příkladu programu Microsoft Word, ve kterém tuto práci edituji (Obrázek 1).



Obrázek 1: Snímek obrazovky aplikace Microsoft Word

Všimněme si tlačítek v panelu nástrojů. Všechna tato tlačítka obsahují jak text, tak obrázek. Můžeme tedy říct, že jsou to tzv. složené widgety, zatímco například zmíněné obrázky jsou widgety jednoduché. Tyto jednoduché widgety jsou závislé na složených widgetech, které jsou jim nadřazené.

Složený widget však nemusí být složený pouze z widgetů jednoduchých. Celý panel nástrojů je totiž také jediný složený widget, který je složen z několika záložek. Každá tato záložka obsahuje spoustu závislých tlačítek, a tato tlačítka obsahují text a obrázek. Celé okno takto vytváří tzv. strom (tedy graf, který neobsahuje cykly – viz [4])

4 PŘÍKLADY KNIHOVEN PRO TVORBU GUI V PYTHONU

Existuje velké množství různých grafických knihoven pro python, přičemž každá má určité výhody i nevýhody. Velká část těchto knihoven je napsaná v jazyce C, případně v assembleru, a to z důvodu rychlosti (viz kapitolu 2). Já sám jsem již s několika grafickými knihovnami pracoval, a právě tyto knihovny nyní krátce ze svého pohledu popíši.

Z následujících jsou knihovny Turtle, Tkinter a Curses součástí standardního balíku knihoven dodávaných s instalací Pythonu. Curses však bohužel nefunguje pod systémem Windows.

4.1 Turtle

Turtle [5] (v překladu želva) je první grafická knihovna, ve které jsem se učil programovat. Je to knihovna primárně určená dětem, aby si mohly vizuálně osahat nejzákladnější principy programování. Turtle umožňuje vytvoření prázdného okna, tzv. plátna, uvnitř kterého se konečnou rychlostí pohybuje želva. Ovládat ji lze jednoduchými příkazy `turtle.forward(20)` (pohyb dopředu 20 pixelů), `turtle.right(35)` (otočení o 35° po směru hodinových ručiček) apod. Tato želva může cestou kreslit čáru, případně může vybarvit útvar ohraničený libovolnou uzavřenou křivkou. Kromě toho je možné odchytávat uživatelem spuštěné události a zpracovávat je. Knihovna umožňuje jak objektově, tak procedurálně orientovaný přístup.

Tato knihovna je však nepoužitelná pro vytváření nějaké trochu složitější aplikace. Neumožňuje kreslení žádných složitějších objektů (např. obdélníků), programátor je tedy nucen si všechno otrocky naprogramovat pomocí pohybů želvy. Samotný pohyb želvy je relativně pomalý. Má samozřejmě nastavitelnou rychlost, jednoduchý objekt je možné vykreslit i prakticky okamžitě; avšak pro renderování celé obrazovky je i želva s neomezenou rychlostí nepoužitelná.

4.2 Tkinter

Tkinter [6] je interface pro knihovnu Tk [7]. která je napsaná původně pro jazyk Tcl. Sám s touto knihovnou nemám příliš mnoho zkušeností, jelikož mě před několika lety úplně nezaujala. Z dnešního pohledu na mě však působí velmi sympaticky. Na internetu je dostupná rozsáhlá dokumentace i samotný zdrojový kód [8].

Tk je vysokoúrovňová knihovna s velmi rozsáhlou databází widgetů. Kromě nejjednodušších widgetů, jako je tlačítko či štítek, implementuje i například menu, combobox, progressbar či „strom“ (treeview). Výše zmíněná knihovna Turtle je na knihovně Tk založená, jedním z dostupných widgetů je tedy i plátno – uvnitř něj poté může kreslit želva. Tkinter má také systém odchytávání a zpracovávání eventů.

4.3 PyQt5

S touto knihovnou jsem se setkal při řešení Korespondenčního semináře z programování [9]. Je to taktéž vysokoúrovňová knihovna, na první pohled programátorsky příjemnější než Tkinter. V současnosti však neexistuje kompletní dokumentace, což je výrazná nevýhoda.

4.4 Curses

Curses [10] je knihovna výjimečná tím, že operuje v prostředí textové konzole, např. v linuxovém terminálu. Přesto však poskytuje relativně vysokoúrovňový interface. Ten neobsahuje tolik různých widgetů jako např. Tkinter, stále je to však výrazná změna, pomocí které je možné vytvořit příjemnou grafickou aplikaci i tehdy, když není možné použít standardní GUI. To může nastat například při spouštění počítače. Se stejnou situací jsem se setkal i při programování robota Breach [11] od společnosti Bender Robotics. Na tomto robotu je plnohodnotný počítač běžící pod linuxem. Pro interakci s robotem je nutno připojit se k počítači pomocí protokolu ssh, tedy přes terminál, který umožňuje jen textovou interakci s robotem.

5 PYGAME

Pygame [12] je nízkoúrovňová knihovna optimalizovaná pro vytváření her. Je postavená na velmi populární knihovně SDL [13]. Níže cituji klíčové vlastnosti pygame.

„Multi core CPUs can be used easily. With dual core CPUs common, and 8 core CPUs cheaply available on desktop systems, making use of multi core CPUs allows you to do more in your game. Selected pygame functions release the dreaded python GIL, which is something you can do from C code.

Uses optimized C and Assembly code for core functions. C code is often 10-20 times faster than python code, and assembly code can easily be 100x or more times faster than python code.

Truly portable. Supports Linux (pygame comes with most main stream linux distributions), Windows (95, 98, ME, 2000, XP, Vista, 64-bit Windows, etc), Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX. The code contains support for AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS and OS/2, but these are not officially supported. You can use it on hand held devices, game consoles and the One Laptop Per Child (OLPC) computer.

It's Simple and easy to use. Kids and adults make shooter games with pygame. Pygame is used in the OLPC project and has been taught in essay courses to young kids and college students. It's also used by people who first programmed in z80 assembler or c64 basic.

You control your main loop. You call pygame functions, they don't call your functions. This gives you greater control when using other libraries, and for different types of programs.

Does not require a GUI to use all functions. You can use pygame from a command line if you want to use it just to process images, get joystick input, or play sounds.

Modular. You can use pieces of pygame separately. Want to use a different sound library? That's fine. Many of the core modules can be initialized and used separately.” [14]

5.1 Surface

Základním objektem v pygame je třída `pygame.Surface`. Je to objekt reprezentující obecně jakýkoli obrázek. Je možné nastavit bitovou hloubku barev, barevnou paletu, barevný klíč, průhlednost a spoustu dalších vlastností. Také je možné načíst obrázky ze souborů nebo uložit surface jako obrázek.

Základní operací, kterou je možno se surface provést, je tzv. blitnutí, tedy zkopírování (části) jednoho surface na druhý. Tímto způsobem probíhá i samotné vykreslování okna. Metoda `pygame.display.set_mode()`, pomocí které se vytváří okno aplikace, taktéž vrací surface. Na tento surface poté můžeme kreslit a všechny změny jsou eventuálně viditelné na displeji. Konkrétnější vysvětlení je možné najít v úvodu k pygame [15], případně v tutoriálu na 2D herní grafiku [16].

5.2 Rect

Dalším důležitým objektem je `pygame.Rect`. Je to velmi obecný a variabilní objekt reprezentující obdélník v souřadnicovém systému. Obdélníky jsou většinou používány k definování určitých oblastí okna, například si mohu pomocí obdélníku uložit, kde je zobrazená hlavní postava hry. Pokud ji poté potřebuji přesunout, vymažu postavu z aktuálních pixelů, posunu obdélník a následně ji nakreslím na nově definovanou oblast.

Obdélník je standardně určen polohou levého horního rohu a rozměry. Má však také několik atributů, které jsou po každé změně přepočítávány (např. topleft, bottomright, center či midright; pro kompletní seznam viz dokumentaci). Je také možné jim přiřadit novou hodnotu, v takovém případě budou všechny ostatní atributy přepočítány. Je-li změněn jakýkoli jiný atribut než některý rozměr obdélníku, obdélník je vždy posunut tak, aby vyhovoval nové hodnotě.

6 PYGAME_WIDGETS

6.1 Použití knihovny

Základní principy použití knihovny pygame_widgets vysvětlím velice podobně, jako jeden z autorů knihovny pygame Pete Shinners v tutoriálu Python Pygame Introduction [17]. Jako příklad uvedu úplně stejný program, avšak naprogramovaný pomocí mé knihovny. Tento program je animací míče odrážejícího se od okrajů okna.

```
1  import pygame_widgets
2
3  size = width, height = 320, 240
4  speed = [2, 2]
5
6  window = pygame_widgets.Window(size, bg_color=(0, 0, 0), fps=0)
7  ball_img = pygame_widgets.pygame.image.load("intro_ball.png")
8  ball = pygame_widgets.Image(window, auto_res=True, image=ball_img)
9
10 while True:
11     window.handle_events(*pygame_widgets.pygame.event.get())
12     ball.move_resize(speed)
13     if ball.master_rect.left <= 0 or ball.master_rect.right >= width:
14         speed[0] *= -1
15     if ball.master_rect.top <=0 or ball.master_rect.bottom >= height:
16         speed[1] *= -1
17     pygame_widgets.new_loop(window)
18
```

Na 1. řádku importujeme knihovnu pygame_widgets. Během importu dojde zároveň k inicializaci pygame, tudíž se o to nemusí uživatel starat.

Na 6. řádku vytváříme grafické okno. To nemá žádné povinné parametry, my však specifikujeme jeho velikost a barvu pozadí a nastavujeme neomezenou snímkovou frekvenci. Poté načítáme obrázek míče, abychom ho mohli použít k vytvoření widgetu míče. Tento widget by měl zobrazit pouze obrázek, vybrali jsme si tedy widget `Image`. Prvním, povinným argumentem specifikujeme, že se má zobrazit uvnitř okna, dále musíme určit, že má zobrazit náš dříve načtený obrázek míče. Pomocí argumentu `auto_res` zajistíme, že bude mít celý widget velikost načteného obrázku, a nemusíme mu ji explicitně říkat.

Řádky 10 až 17 tvoří tzv. mainloop neboli hlavní smyčku. Ta je vykonávána opakovaně až do ukončení programu, a stará se o zpracovávání událostí (tzv. eventů), které mohl operační systém nebo uživatel vyvolat, a o aktualizaci obrazu na displeji. To je pro nás v tomto případě nejzásadnější. Většina ostatních knihoven funguje tak, že je hlavní smyčka někde interně naprogramovaná a programátor k ní nemá přístup – pouze ji spustí například příkazem `window.mainloop()`.

To, že máme plně pod kontrolou hlavní smyčku, je klíčová vlastnost pygame. Vnímám to jako velkou výhodu, je s tím však spojena i určitá zodpovědnost. Každá aplikace by totiž měla správným způsobem reagovat na některé systémové eventy (patří mezi ně například i kliknutí na červený křížek), jinak ji může operační systém vyhodnotit jako zamrzlou a násilně ji ukončit. Tyto povinné kroky, které se musí během každé iterace hlavní smyčky provést, jsou obsaženy v 11. a 17. řádku naší animace.

Na 11. řádku načítáme pomocí standartní funkce z pygame všechny eventy, které byly vyvolány od posledního volání. Avšak místo toho, abychom přes ně ručně iterovali a pro každý se rozhodovali pomocí spousty podmínek, jak na něj zareagujeme, pouze necháme okno, aby je

zpracovalo. To je rozešle všem ostatním widgetům (v tomto příkladě pouze našemu obrázku) a všechny widgety na ně mohou zareagovat. Děje se to pomocí tzv. handlerů, což jsou v podstatě obyčejné funkce (případně metody). Každý widget může mít sadu handlerů asociovaných s určitým eventem. Některé widgety mají nastavené handlery už při inicializaci (například `pygame_widgets.Window` reaguje na event `pygame.QUIT` tak, že ukončí program). Handlery můžeme přidávat, odebírat a měnit jejich pořadí pomocí několika metod uvedených v dokumentaci. V našem příkladě však tuto možnost nevyužíváme.

Funkce `pygame_widgets.new_loop()` na 17. řádku je podrobně vysvětlena v dokumentaci. Prozatím postačí, že uvnitř této funkce dochází k samotné aktualizaci displeje.

Zbytek kódu v hlavní smyčce je velmi podobný kódu použitého v originálním programu. Na 12. řádku přesuneme widget podle aktuální rychlosti relativně vůči jeho poloze (viz dokumentaci `pygame_widgets.widgets.widget._Widget.move_resize()`). Je dobré si uvědomit, že tato metoda se již sama stará o vymazání původního obrazu, přesunutí widgetu a jeho následné vykreslení, zatímco v originálním programu jsme museli původní obraz ručně vymazat vyplněním celého okna. To by byl velký rozdíl v nějaké složitější aplikaci. Představme si situaci, kdy potřebujeme zobrazit další, avšak nehybný balón někde uvnitř okna. Vyplnění obrazovky by ho v každém snímku vymazalo, tudíž bychom ho museli vykreslovat pokaždé znovu. Knihovna pygame_widgets vymaže vždy jen tu část obrazovky, kde se widget původně nacházel. Také se interně postará o to, aby se správně zobrazily widgety, které byly předtím částečně či úplně skryté.

Následně kontrolujeme, jestli se míč odrazil od okraje okna. Atribut `ball.master_rect` je obdélník udávající rozměry a polohu míče uvnitř přímo závislého widgetu. Tím je v tomto případě okno.

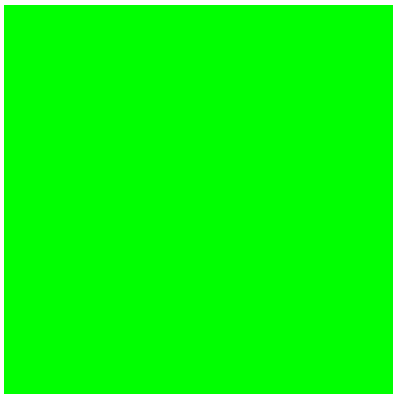
Tím bych uzavřel tento krátký úvod.

6.2 Interní struktura knihovny

Pro pochopení interní struktury knihovny je nutné plně ovládat metodu `pygame.Surface.subsurface()`. Tu nyní vysvětlím na dalším příkladu.

```
1  import pygame
2  pygame.init()
3  surf_parent = pygame.display.set_mode((400, 400))
4  surf_parent.fill((0, 255, 0))
5  pygame.display.flip()
```

Nejdříve importuji a inicializuji pygame, poté vytvořím okno s rozměry 400×400 pixelů. Metoda `pygame.display.set_mode()` vrátí surface, který bude okno reprezentovat. Jestliže budeme chtít něco nakreslit na okno, nakreslíme to na `surf_parent`. Na dalším řádku vyplníme `surf_parent` zelenou barvou a po zavolání `pygame.display.flip()` se tato změna objeví na displeji (viz Obrázek 2)

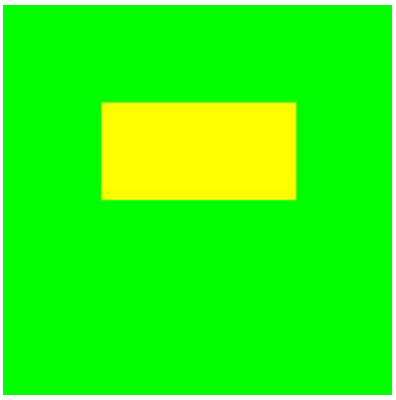


Obrázek 2: Snímek obrazovky 1

Nyní k programu přidám další 4 řádky kódu:

```
6  rect = pygame.Rect(100, 100, 200, 100)
7  surf_child = surf_parent.subsurface(rect)
8  surf_child.fill((255, 255, 0))
9  pygame.display.flip()
```

Vytvářím obdélník s rozměry 200×100, jehož levý horní roh má souřadnice $(100; 100)$. Na dalším řádku ho použiji k vytvoření výřezu obrazovky – `surf_child` je surface s rozměry 200×100 pixelů, který sdílí své pixely se `surf_parent`. Nakreslíme-li něco na jeden z těchto surface, změna se objeví i na druhém z nich (pokud půjde samozřejmě v případě `surf_parent` o oblast definovanou obdélníkem `rect`). Následné 2 řádky to demonstrují. Nejdříve totiž vyplníme `surf_child` žlutou barvou a poté aktualizujeme displej. Na obrazovce tedy vidíme oblast, která je sdílená mezi rodičem a dítětem (Obrázek 3).



Obrázek 3: Snímek obrazovky 2

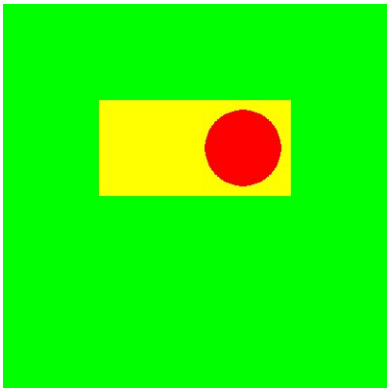
Jestliže nyní chceme něco (například červený kruh) nakreslit do žlutého obdélníku, máme 2 možnosti. Můžeme použít původní surface:

```
10 pygame.draw.circle(surf_parent, (255, 0, 0), (250, 150), 40)
11 pygame.display.flip()
```

Nebo můžeme kreslit na `surf_child`:

```
10 pygame.draw.circle(surf_child, (255, 0, 0), (150, 50), 40)
11 pygame.display.flip()
```

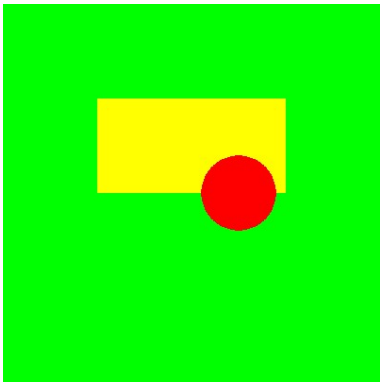
Oběma způsoby získáme stejný výsledek na obrazovce (Obrázek 4), avšak pokaždé jsme použili jiné souřadnice středu kruhu (3. argument). Počátkem soustavy souřadnic je totiž vždy horní levý roh surface, na který se kreslí.



Obrázek 4: Snímek obrazovky 3

Nyní se zamyslíme nad otázkou, co se stane, když červený kruh posuneme o 50 pixelů dolů. V tu chvíli by měl částečně zasahovat do žluté a částečně do zelené oblasti. Opravdu tomu tak bude (Obrázek 5), když modifikujeme první případ:

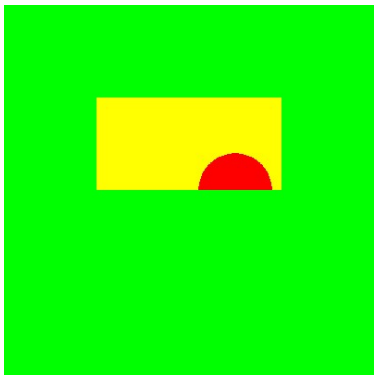
```
10 pygame.draw.circle(surf_parent, (255, 0, 0), (250, 200), 40)
11 pygame.display.flip()
```



Obrázek 5: Snímek obrazovky 4

Ve druhém případě se nám ovšem vykreslí jen horní polovina kruhu, ta, která je uvnitř sdílené oblasti (Obrázek 6):

```
10 pygame.draw.circle(surf_child, (255, 0, 0), (150, 100), 40)
11 pygame.display.flip()
```



Obrázek 6: Snímek obrazovky 5

Pomocí subsurface můžeme ovlivnit jen ty pixely, které jsou ve sdílené oblasti dítěte a rodiče.

Této funkcionality jsem využil pro tvorbu widgetů v pygame_widgets. Každý widget (tj. instance třídy dědící z `pygame_widgets.widgets.widget._Master`) má atribut `widget.surface`. To je reference na tu část obrazovky, kterou má widget k dispozici. V případě `pygame_widgets.Window` je to surface odpovídající celému oknu (tedy návratová hodnota `pygame.display.set_mode()`), v případě ostatních widgetů je to jeho subsurface.

Pokaždé, když inicializujeme nějaký závislý widget, prvním, povinným argumentem je master. Je to reference na nějaký již existující widget. Nový widget se poté připojí do stromu existujících widgetů jako přímý potomek widgetu `master`. Znamená to, že `widget.surface` bude potomkem `master.surface`, tedy může být widget zobrazen pouze uvnitř prostoru, který má k dispozici `master`. Tento argument je poté uložen v atributu `widget.master`.

Kromě toho má každý widget atribut `widget.my_surf`. V tomto surface je uložen aktuální vzhled widgetu, který je posléze kopírován na `widget.surface` (čímž se objeví na displeji). Toto je zavedeno z toho důvodu, že `widget.surface` může být kdykoli překresleno pomocí kteréhokoli surface, který vlastní referenci na některé z pixelů `widget.surface`. V tu chvíli by byly ztracené některé informace o vzhledu, které by bylo poté nutné generovat z nastavení widgetu. To by bylo časově neefektivní.

Samotné vytváření vzhledu widgetu (tedy vykreslování `widget.my_surf`) probíhá uvnitř metody `widget._generate_surf()`. Tato metoda je privátní, proto není zmíněna v dokumentaci; je však důležité o ní vědět v případě vytváření vlastních widgetů.

Pozici závislého widgetu uvnitř nadřazeného a jeho velikost určuje atribut `widget.master_rect`, instance třídy `pygame.Rect()`. Jeho souřadnice je relativní vůči levému hornímu rohu přímo nadřazeného widgetu, tedy vůči `widget.master.master_rect.topleft`. Obdélník `master_rect` však nemusí nutně být kompletně uvnitř nadřazeného widgetu, může do něj zasahovat například jen částečně nebo vůbec. V takovém případě ovšem bude widget částečně či úplně skrytý z důvodů demonstrovaných v předchozím příkladu.

V prvních verzích knihovny vždy platilo, že `widget.surface` má vždy stejnou velikost i pozici jako `widget.master_rect`. To se však později ukázalo jako nevyhovující, právě protože nebylo možné, aby byl widget mimo nadřazený widget. Argument metody `surface.subsurface` totiž musí být vždy kompletně uvnitř `surface`. Řešením je, že `widget.surface` je nyní vždy průnikem `widget.master_rect` se surface jeho nadřazeného widgetu. Pro správné vykreslování bylo poté nutné přidat atribut `widget.topleft`, který určuje souřadnice levého horního rohu `widget.master_rect` relativně k levému hornímu rohu `widget.surface`.

Zmiňované atributy `master`, `master_rect`, `surface`, `my_surf` a `topleft` slouží pouze pro čtení. Není implementována žádná ochrana proti zápisu, vedlo by to však ke kolapsu knihovny. Nějaký způsob ochrany bude pravděpodobně implementován v blízké budoucnosti.

6.3 Výhody a nevýhody

Hlavní výhodou, na kterou jsem se při psaní knihovny velice soustředil, je vysoká variabilita. Chtěl jsem zajistit velkou volnost programátora při vytváření aplikací, tato volnost mi chyběla při programování pomocí jiných grafických knihoven. Zároveň jsem však chtěl, aby knihovna za programátora vyřešila co nejvíce detailů, pokud je programátor nechce řešit sám. Často jsem tuto situaci vyřešil tím způsobem, že programátor může nahradit část kódu svým vlastním. Mohu to ilustrovat na příkladu vzhledu tlačítka: widget `pygame.Button` lze jednoduše a velmi rychle instancovat pouze s pojmenovaným argumentem `text`, čímž se vytvoří funkční tlačítko s defaultním vzhledem (Obrázek 7). Chce-li však programátor kontrolovat i vzhled, má spoustu různě náročných možností. V opačném extrému může tlačítko vypadat třeba jako na Obrázek 8 (Tento obrázek je výřez obrazovky ze hry Heroes of Might and Magic III [18])

Obrázek 7: Výchozí vzhled tlačítka

Obrázek 8: Přizpůsobený vzhled tlačítka

K tomuto účelu si může programátor naprogramovat vlastní funkci, která bude vytvářet nějaký komplikovaný vzhled, a poté ji použít k nahrazení defaultní funkce pro vytváření vzhledu (viz dokumentaci `pygame_widgets.Button.set()`).

Další výhodou je podle mého soudu vyčerpávající dokumentace, která je součástí této práce. Dokumentace v českém jazyce není příliš užitečná, v nejbližší budoucnosti však bude i

dokumentace v anglickém jazyce dostupná v repozitáři https://github.com/Jajasek/pygame_widgets společně se zdrojovým kódem knihovny.

Nejzásadnější nevýhodou mé knihovny je nízká rychlost. Ta je ovlivněna hlavně faktem, že jsem celou knihovny napsal v Pythonu, avšak kromě toho je také velmi špatně optimalizovaná. Je to dáno tím, že jsem se v průběhu programování teprve spoustu věcí učil.

6.4 Způsob sdílení

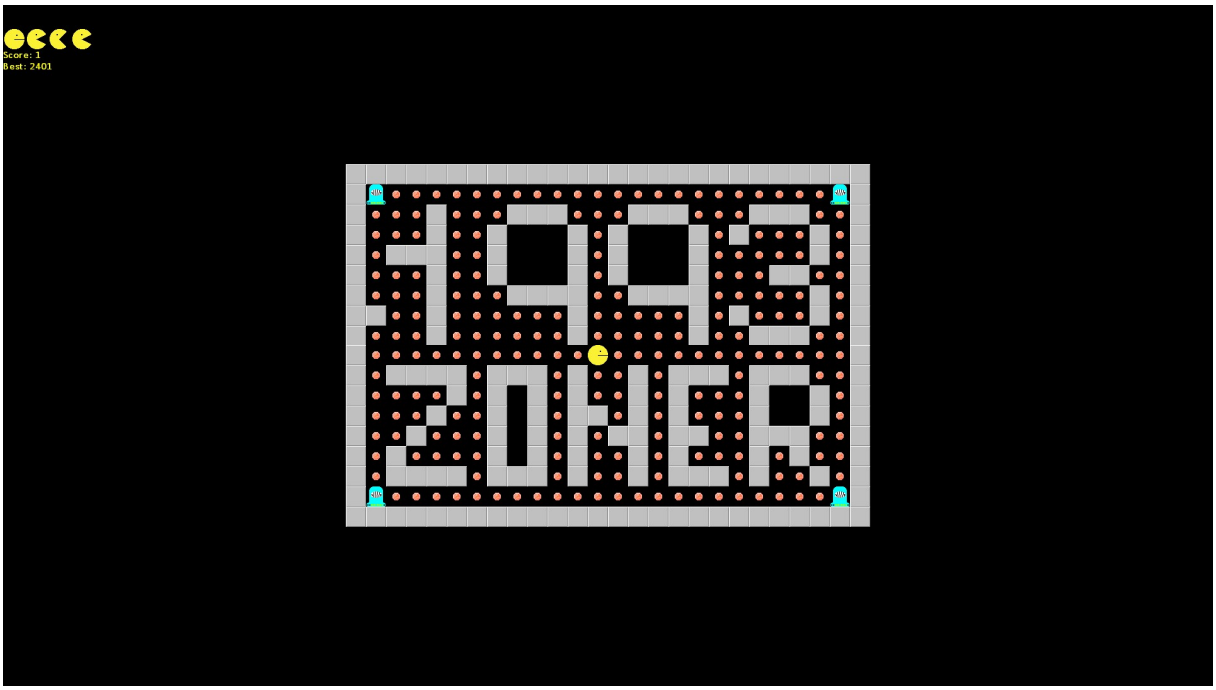
Knihovna pygame_widgets je open source projekt. Znamená to, že každý má zdarma přístup ke zdrojovému kódu a právo ho jakkoli používat, šířit či editovat. Snažím se tím usnadnit přístup potenciálních uživatelů ke kódu. Ten je verzován pomocí systému Git a publikován v repozitáři https://github.com/Jajasek/pygame_widgets.

6.5 Pampuch

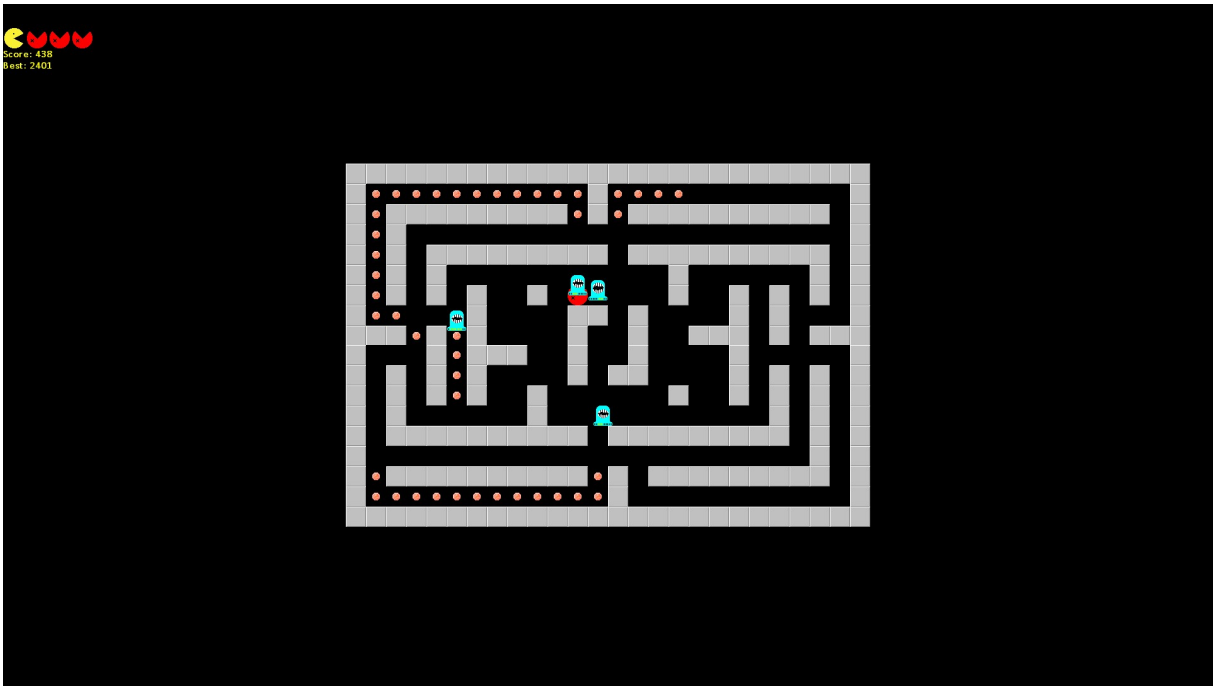
Svou knihovnu jsem již použil pro reimplementaci hry Pampuch, vydané roku 1994 společností ZONER software [19]. Tato hra je volně přístupná pro operační systém Windows XP, neexistovala však funkční verze pro Windows 10. Z toho důvodu jsem vytvořil její kopii právě pro tento operační systém, teoreticky by však měla být funkční na jakémkoli systému podporující pygame. Zdrojový kód je open source a je dostupný v repozitáři https://github.com/Jajasek/Pampuch.

Pampuch je jedna z mnoha verzí populární hry Pacman, má však několik specifických vlastností. Hlavní rozdíl je v implementaci umělé inteligence ovládající 4 „duchy“. Ta je velmi predikovatelná, což přidává do hry velice zábavnou strategickou rovinu. „Duchové“ mění směr pohybu pouze tehdy, narazí-li do zdi či do jiného „ducha“, tvar bludiště tedy zásadním způsobem ovlivní obtížnost a strategii dané úrovně.

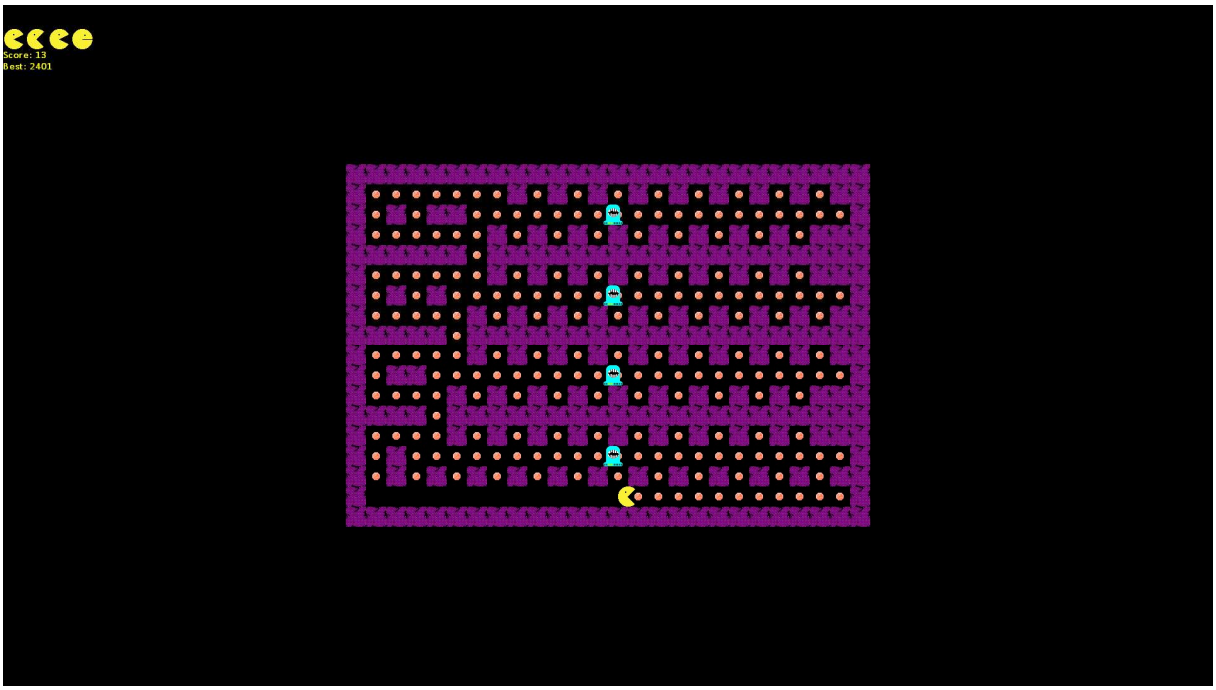
Nyní následuje několik snímků obrazovky.



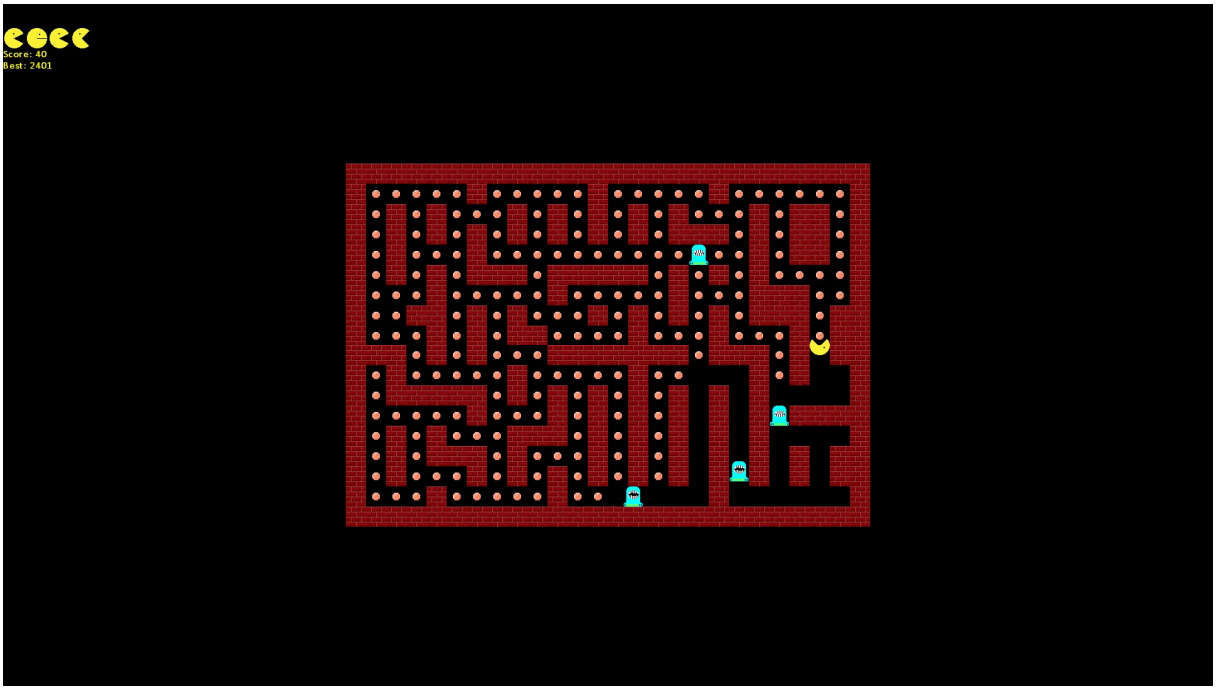
Obrázek 9: Snímek obrazovky hry Pampuch, úroveň 1



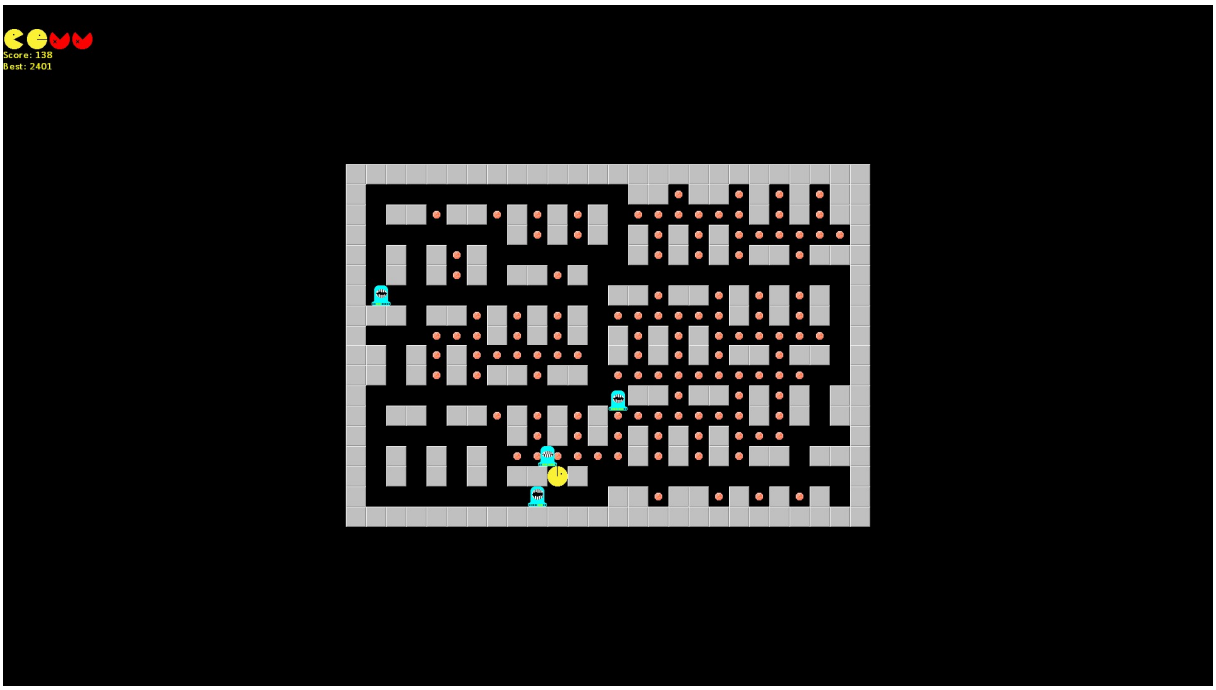
Obrázek 10: Snímek obrazovky hry Pampuch, úroveň 2



Obrázek 11: Snímek obrazovky hry Pampuch, úroveň 4



Obrázek 12: Snímek obrazovky hry Pampuch, úroveň 7



Obrázek 13: Snímek obrazovky hry Pampuch, úroveň 10

6.6 Dokumentace modulů, objektů a jejich metod

Tato dokumentace je aktuální k 17. 3. 2020.

`pygame_widgets`

kořenový adresář knihovny

Kořenový adresář (lépe řečeno skript `pygame_widgets\__init__.py`) obsahuje referenci na importovanou knihovnu pygame: `pygame_widgets.pygame`. Je však možno pygame importovat zvlášť. Stojí za zmínku, že v takovém případě není nutné volat `pygame.init()`, jelikož pygame byl již inicializován při importu pygame_widgets.

`new_loop(window=None) -> None`

oznámit další iteraci hlavní smyčky

Uvnitř této metody jsou zabalené různé věci, které je nutno v každé iteraci hlavní smyčky provést. Primárně dochází k publikaci eventu `E_LOOP_STARTED`, dále se tato funkce stará o zpožděné volání funkcí pomocí `delayed_call()`. Jestliže je argument `window` instance třídy `pygame_widgets.Window` (či z ní dědí), pak je aktualizovaná obrazovka pomocí `Window.update_display()`.

Je silně doporučeno zavolat `new_loop()` právě jednou v každé iteraci hlavní smyčky programu, jelikož event `E_LOOP_STARTED` může být vyžadován pro správnou funkčnost knihovny.

`delayed_call(func, delay=1, *args, **kwargs) -> None`

zavolat funkci se zpožděním

Tato funkce umožňuje zavolat `func(*args, **kwargs)` po uplynutí tolika iterací hlavní smyčky, kolik je argument `delay`. Knihovna tuto situaci pozná pomocí volání funkce `new_loop()`, uvnitř této funkce také dochází ke zpožděnému volání `func`.

`set_mode_init() -> None`

zabrání widgetům vytvářet eventy

Tato funkce byla implementována po několika neúspěšných testech, při kterých moje knihovna zaplnila frontu eventů a následně spadla. Při inicializaci velkého množství widgetů dochází totiž k publikaci mnoha eventů, přičemž se všechny ve frontě kumulují až do spuštění hlavní smyčky – teprve tehdy dojde k jejich postupnému zpracovávání. Tyto eventy přitom nejsou potřeba – ohlašují změny atributů widgetů, které však při inicializaci očekáváme.

`set_mode_mainloop() -> None`

opět povolit publikaci eventů

Po zavolání `set_mode_init()` je možné takto opět povolit widgetům generovat eventy.

`get_mode() -> None`

vrátí `True`***, jestliže mají widgety povoleno publikovat eventy, jinak*** `False`

`pygame_widgets.widgets.widget._Master`

objekt, který obsahuje metody společné všem widgetům

Objekt není možné instancovat ani z něj dědit.

`on_screen(rect=None) -> bool`

vrátí `False`***, pokud je*** `rect` ***kompletně mimo zobrazovanou oblast, jinak*** `True`

Pokud není vložen argument `rect`, použije se `self.get_abs_master_rect()`.

`kwarg_list() -> list`

vrátí seznam všech nastavitelných atributů

Prvky seznamu jsou řetězce se všemi klíčovými slovy, které akceptuje metoda `set()`.

```
add_handler(event_type, func, args=None, kwargs=None,
            self_arg=True, event_arg=True, delay=0, index=None)
            -> None
```

přidá eventům určitého typu nový handler s danými vlastnostmi

`event_type` je typ zpracovávaných eventů, `func` je samotný handler, `args` je iterovatelný objekt s polohovými argumenty pro `func`, `kwargs` je slovník (či jakýkoli mapping) s pojmenovanými argumenty. `self_arg` a `event_arg` jsou boolovské argumenty, které říkají, jestli má funkce `func` při volání převzít odkaz na widget, který handler spustil, a na event, který je zpracováván. Tyto argumenty se případně vloží na začátek seznamu `args` (`self` bude vždy první). Argument `delay` je nezáporné celé číslo určující počet programových cyklů, které mají proběhnout, než se handler spustí. Toho je možné využít k lepší kontrole pořadí spouštěných handlerů. K podobnému účelu slouží `index`, ten specifikuje, na jaké místo v seznamu handlerů má být nový handler vložen (používá metodu `list.insert()`). Není-li `index` vložen, handler se přidá na konec seznamu.

Kontrola pořadí je však stále velmi provizorní, proto bude v budoucnosti pravděpodobně vyřešena jinak.

```
remove_handler(event_type, func, args=None, kwargs=None,
               self_arg=True, event_arg=True) -> None
```

Odstraní všechny handlery s danými vlastnostmi

Aby byl handler odstraněn, musí se všechny argumenty rovnat argumentům, se kterými byl vytvořen.

```
get_handlers(copy=True) -> dict
```

Vrátí slovník se všemi handlery daného widgetu

Klíče tohoto slovníku jsou jednotlivé typy eventů a odpovídající hodnoty jsou seznamy handlerů, které jsou volány (v tomto pořadí) při zpracovávání eventů tohoto typu. Handlery jsou instance třídy `pygame_widgets.handler.Handler`.

Je-li `copy == False`, pak je vrácený slovník referencí na ten slovník, ve kterém jsou handlery v daném widgetu uloženy. Uživatel tedy může tento slovník měnit. V takovém případě má absolutní kontrolu nad pořadím handlerů, zároveň však může poškodit základní funkce widgetu. Tato možnost je určena jen pro zkušeného uživatele, který ví, co dělá.

V opačném případě je celý slovník včetně všech seznamů a handlerů pouze kopií a změny, které na něm uživatel provede, se nijak neprojeví na chování widgetu.

```
add_grab(event_type, child, level=0) -> None
```

přeposílat všechny eventy typu `event_type` ***pouze widgetu*** `child`

Tato metoda má využití například při vytváření textového zadávacího pole. Jestliže do něj klikneme, požadujeme, aby ostatní widgety nereagovaly na stisknuté klávesy neboli aby nezískávaly informace o eventech typu `pygame_widgets.KEYDOWN`. Toho je docíleno příkazem

```
entry.master.add_grab(pygame_widgets.KEYDOWN, entry, -1)
```

Tato metoda se poté spouští rekurzivně, tj. přímo nadřazený widget tohoto zadávacího pole také zamkne eventy `pygame_widgets.KEYDOWN` jen pro sebe, aby je mohl přeposílat dál. Argument `level` určuje hloubku této rekurze. Je-li to záporné číslo, rekurze proběhne až k absolutně nadřazenému widgetu.

`remove_grab(event_type, child, level=0) -> None`

zrušit účinky `add_grab()`

Argumenty mají stejný význam, jako u metody `add_grab()`.

`add_nr_events(*args) -> None; remove_nr_events(*args) -> None`

přidat nebo odebrat typy eventů, které bude widget ignorovat

Widget nebude tyto eventy zpracovávat, ani je přeposílat závislým widgetům. Typy eventů jsou uloženy v množině, proto je bezpečné přidat jeden typ eventu vícekrát.

`add_ns_events(*args) -> None; remove_nr_events(*args) -> None`

přidat nebo odebrat typy eventů, které nebude widget přeposílat

Velmi podobné metodám `add_nr_events()` a `remove_nr_events()` s tím rozdílem, že widget bude eventy těchto typů zpracovávat, pouze je nebude přeposílat závislým widgetům.

`handle_event(event, _filter=True) -> None`

zpracovat event a přeposlat ho závislým widgetům

event je instance třídy `pygame.event.Event`. Jestliže byl vygenerovaný nějakým widgetem, tento widget ho zároveň v okamžiku vygenerování i zpracoval. Aby nebyl event zpracován tímto widgetem dvakrát, má atribut `event.widget`, kde je uložena reference na daný objekt. Widget poté nezpracovává ty eventy, které mají jeho podpis, pouze je přepošle závislým widgetům.

Je-li `_filter == False`, pak je ale toto filtrování vypnuté. To je využíváno interně uvnitř knihovny, uživatel by však měl vždy nechat tento argument na výchozí hodnotě.

`handle_events(*events, _filter=True) -> None`

postupně zpracovat všechny zadané eventy a přeposlat je závislým widgetům

Tato metoda zpracuje každý event pomocí metody `handle_event()`.

`blit(rect=None, _update=True) -> None`

Překreslí část displeje

Používá metodu `pygame.surface.blit()`. Blitne část `self.my_surf` na `self.surface`, a tím i na `Window.surface`. Oblast blitnutí je určena argumentem `rect`, což je instance třídy `pygame.Rect`. Není-li zadán, použije se `self.surface.get_rect()`. Souřadnice jsou počítány relativně k levému hornímu rohu `self.master_rect`.

Tato metoda je rekurzivně spuštěna na všech závislých widgetech, které zasahují do daného obdélníku. Kvůli optimalizaci je při tom nastaven argument `_update` na `False`, uživatel by ho však měl vždy nechat na výchozí hodnotě.

`pygame_widgets.Window`

widget, který představuje celé okno aplikace; jediný nezávislý widget

```
Window(resolution=(0, 0), flags=0, depth=0, **kwargs) -> Window instance
```

Pomocí objektu `Window` je možné ovládat základní funkce aplikace. Jakmile je vytvořena instance, zobrazí se okno a je možné na něm zobrazit další widgety. Okno se také stará o zpracovávání eventů a jejich distribuci ostatním widgetům.

Argument `resolution` udává rozlišení okna. Pokud je některý z rozměrů 0, tento rozměr se nastaví na odpovídající rozměr rozlišení displeje. Argumenty `flags` a `depth` mají stejný význam, jako stejnojmenné argumenty pro funkci `pygame.display.set_mode()`. Obvykle je nejlepší nechat `depth` na výchozí hodnotě, protože poté ji pygame nastaví na nejlepší hodnotu pro daný systém. Argument `flags` mění vlastnosti okna.

Pojmenované argumenty slouží k nastavení okna okamžitě po inicializaci. Slouží to jen k usnadnění. Následující dva bloky kódu mají naprosto stejný výsledek:

```
window = pygame_widgets.Window(**kwargs)

window = pygame_widgets.Window()
window.set(**kwargs)
```

Popis jednotlivých pojmenovaných argumentů je uveden v dokumentaci metody `Window.set()`.

Tato třída dědí ze třídy `_Master`.

Upozornění: Pygame umožňuje zobrazení pouze jednoho okna, při vytvoření více instancí objektu `Window` se bude aplikace chovat nepředvídatelně.

`set(**kwargs) -> None`

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `Window`, podporují všechny argumenty, které podporuje `Window`.

Po každý nastavený atribut publikuje event `E_WINDOW_ATTR`, který má atributy `name` (string obsahující jméno atributu), `new` (nově přiřazená hodnota) a `old` (původní hodnota; pro některé atributy rovna `None`).

`cursor`	výchozí vzhled kurzoru
`fps`	maximální snímková frekvence (používá `pygame.time.Clock`). Je-li `fps==0`, snímková frekvence není omezena.
`bg_color`	barva pozadí
`min_size, max_size`	určuje minimální a maximální velikost okna, pokud má uživatel možnost velikost měnit (je povolena možnost `RESIZABLE`) nebo se ji snaží měnit sám program.
`title`	nastaví titulek, který se zobrazí v rámečku okna
`icontitle`	nastaví titulek, který se zobrazí na hlavním panelu. Je-li nastaveno na `None`, použije se `title`.
`icon`	nastaví ikonu použitou na hlavním panelu
`size`	změní rozlišení okna

`quit(code=0) -> None`

ukončí aplikaci

Deinicializuje všechny aktivní widgety a poté pygame. Následně zavolá `sys.exit(code)`.

`delete() -> None`

rekurzivně deinicializuje všechny aktivní widgety

`update_display() -> None`

aktualizuje obraz na monitoru

Nakreslí na monitor všechny změny od posledního volání. Zároveň aktualizuje hodiny (viz dokumentaci `pygame.time.Clock.tick`). Tato metoda by měla být volána v každém cyklu hlavní programové smyčky, aby byly na monitoru zobrazeny aktuální informace. Je-li nastavené omezení snímkové frekvence, `update_display()` pokaždé počká, aby nebyla překročena.

Při volání funkce `pygame_widgets.new_loop(window)` s platným argumentem `window` se uvnitř této funkce zavolá `window.update_display()`. Jelikož je

doporučeno zavolat funkci `pygame_widgets.new_loop()` v každém cyklu hlavní programové smyčky, nemusí uživatel volat `update_display()` zvlášť.

`get_fps() -> float`

vrátí aktuální snímkovou frekvenci

Spočítá průměrnou snímkovou frekvenci za posledních 10 volání `update_display()`. Používá `pygame.time.Clock.get_fps()`.

`get_visibility() -> True`

vrátí `True`

Tato metoda je pro uživatele naprosto zbytečná, je implementována z důvodu funkčnosti interní mechaniky knihovny. Pravděpodobně bude v budoucnosti změněna.

`add_update(rect=None) -> None`

uloží danou část displeje a překreslí ji při příštím zavolání `update_display()`

Metoda `update_display()` používá pro překreslování funkci `pygame.display.update()`, která je optimalizovaná pro hromadné překreslování více částí obrazovky. Proto knihovna v průběhu hlavní smyčky pouze ukládá obdélníky, které je potřeba překreslit. To poté musí uživatel zařídit pomocí `update_display()`.

Je-li ponechána výchozí hodnota argumentu `rect`, do seznamu se přidá obdélník obsahující celé okno.

```
change_surface(surf, dest=(0, 0), area=None, special_flags=0)
               -> None
```

Změní část pozadí okna

Tato metoda je analogická metodě `pygame.Surface.blit()`.

`pygame_widgets.widgets.widget._Widget`

objekt pro zajištění funkcionality závislých widgetů

`_Widget` by neměl být instancován. Každý widget z něj však dědí, protože `_Widget` obsahuje metody pro zajištění komunikace s `Window` a vykreslování.

Tato třída dědí ze třídy `_Master`.

`set(**kwargs) -> None`

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `_Widgetu`, podporují všechny argumenty, které podporuje `_Widget`.

Po každý nastavený atribut publikuje event `E_WIDGET_ATTR`, který má atributy `name` (string obsahující jméno atributu), `new` (nově přiřazená hodnota) a `old` (původní hodnota; pro některé atributy rovna `None`).

`cursor`	vzhled kurzoru nad widgetem
`auto_res`	toto nastavení zatím ovlivňuje pouze chování pygame_widgets.Label a pygame_widgets.Image. Je-li auto_res == True, velikost labelu a obrázku se mění podle zobrazovaného textu, respektive obrázku.
`visible`	Umožní zneviditelnění nebo opětovné zviditelnění widgetu a všech widgetů na něm závislých. Neviditelný widget stále přijímá eventy.

`delete() -> None`

deinicializuje widget

Rekurzivně zavolá `delete()` na všech přímo závislých widgetech. Poté zavolá `self.disconnect()` a odstraní všechny vazby na přímo nadřazený widget. Při odstranění všech referencí bude widget eventuelně odstraněn garbage collectorem.

`get_abs_master_rect() -> Rect`

vrátí obdélník největšího možného použitého místa v absolutně nadřazeném widgetu

Tento obdélník má rozměry obdélníku `self.master_rect` a souřadnice levého horního rohu widgetu vůči levému hornímu rohu okna. Reprezentuje největší místo, které může widget použít, má-li možnost. Může se však stát, že bude widget částečně či úplně mimo některý ze svých nadřazených widgetů, tedy bude částečně, respektive úplně skrytý. V takovém případě bude reálně využitá plocha menší než návratová hodnota `self.get_abs_master_rect()`. Viz `get_abs_surf_rect()`.

`get_abs_surf_rect() -> Rect`

vrátí obdélník reálně použitého místa v absolutně nadřazeném widgetu

Návratový obdélník je vždy podmnožinou `self.get_abs_master_rect()`. V případě, že widget není celý viditelný (viz dokumentaci `get_abs_master_rect()`), tato metoda vrátí pouze oblast, kde je widget reálně zobrazen. V případě, že je widget kompletně skrytý, vrátí obdélník s nulovými rozměry.

`get_abs_master_path() -> list`

vrátí seznam referencí na všechny nadřazené widgety

Seznam je řazen od absolutně nadřazeného k přímo nadřazenému widgetu.

`get_visibility() -> bool`

vrátí `True`***, je-li viditelný*** `self` ***i každý z nadřazených widgetů***

`appear() -> None`

nakreslí widget na obrazovku a přidá update

`disappear() -> None`

vymaže widget z obrazovky a přidá update

Widget je překreslen ostatními widgety.

`add_update(rect=None) -> None`

uloží danou část displeje a překreslí ji při příštím zavolání `Window.update_display()`

Rekurzivně zavolá `self.master.add_update()`. Viz dokumentaci `Window.add_update()`.

Souřadnice obdélníku `rect` jsou vztaženy k levému hornímu rohu okna. Není-li `rect` zadán, použije se `self.get_abs_master_rect()`.

`disconnect() -> None`

odpojí se od přímo nadřazeného widgetu

Widget se odpojí ze stromu widgetů. Nebude tedy zobrazen na obrazovce ani přijímat eventy. To platí i pro widgety na něm závislé (ty na něm závislé zůstanou). Může být kdykoliv opět připojen pomocí metody `reconnect()`.

`reconnect() -> None`

metoda určená k aktualizaci spojení s přímo nadřazeným widgetem

Tato metoda se pokusí obnovit spojení `self.surface` a `self.master.surface`. Byl-li widget odpojen pomocí metody `disconnect()`, je takto možné opět ho připojit do stromu aktivních widgetů.

```
move_resize(move=(0, 0), move_level=0, resize=(1, 1),
            resize_rel=True, update_surf=True) -> None
```

změní velikost a pozici widgetu v rámci přímo nadřazeného widgetu

`move` udává souřadnice pohybu. Ty mohou být vztaženy k levému hornímu rohu libovolného z nadřazených widgetů, což je určeno argumentem `move_level`. Je-li roven 0, souřadnice jsou relativní (vztaženy k `self.master_rect.topleft`), pro

1 je počítáno s přímo nadřazeným widgetem atd. Je-li `move_level` větší než počet nadřazených widgetů, počítá se s absolutními souřadnicemi v rámci okna.

`pygame_widgets.widgets.text._Text`

virtuální bázový objekt pro textové widgety

Z tohoto objektu dědí všechny widgety, které nějakým způsobem obsahují text. `_Text` totiž implementuje nastavování atributů textu pomocí metody `set()` a následnou aktualizaci vzhledu. Není jej však možno instancovat, protože žádný vlastní vzhled nemá.

Tato třída dědí ze třídy `_Widget`.

`set(**kwargs) -> None`

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `_Text`, podporují všechny argumenty, které podporuje `_Text`.

`font_name`	string obsahující jméno fontu
`font_size`	celé číslo určující výšku textu v pixelech
`bold`	tučné písmo
`italic`	kurzíva
`underlined`	podtržení
`font_color`	barva písma
`smooth`	rozostření okrajů textu (viz argument antialias metody pygame.font.Font.render())
`text`	string, který se má zobrazit
`alignment_x`	vodorovné zarovnání textu, jedna z konstant A_TOPLEFT, A_CENTER a A_BOTTOMRIGHT
`alignment_y`	svislé zarovnání textu, jedna z konstant A_TOPLEFT, A_CENTER a A_BOTTOMRIGHT

`pygame_widgets.Label`

štítek zobrazující jednořádkový text

```
Label(master, topleft=(0, 0), size=(1, 1), **kwargs) -> Label instance
```

`master` je reference na widget, na kterém má být nový label přímo závislý, `topleft` je souřadnice levého horního rohu labelu vůči levému hornímu rohu přímo nadřazeného widgetu, `size` je velikost labelu v pixelech.

Tato třída dědí ze třídy `_Text`.

`set(**kwargs) -> None`

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `Labelu`, podporují všechny argumenty, které podporuje `Label`.

Po každý nastavený atribut publikuje event `E_LABEL_ATTR`, který má atributy `name` (string obsahující jméno atributu), `new` (nově přiřazená hodnota) a `old` (původní hodnota; pro některé atributy rovna `None`). Výjimkou je atribut `text` (zděděný ze třídy `_Text`), jehož změna vyvolá event `E_LABEL_TEXT`.

Label má jediný vlastní pojmenovaný argument `background`, který určuje pozadí widgetu. Pozadí může být definováno buď jednolitou barvou (uspořádaná trojice či čtveřice složek `red`, `green`, `blue`, případně `alpha`, nebo instance třídy `pygame.Color`), objektem `pygame.Surface` (ten bude upraven pomocí `pygame.transform.scale()` na velikost labelu), či libovolným objektem, který lze zavolat. Ten by měl přijmout 2 polohové argumenty, a to `self` (referenci na daný label) a `text` (surface s vyrenderovaným textem). Vrátit by měl hotový surface, který bude přímo zobrazen na obrazovce. Label bude mít velikost vráceného surface.

`pygame_widgets.widgets.button._Button`

virtuální bázový widget, který implementuje detekci stisku tlačítka

Tento objekt obsahuje metody, které detekují stisk tlačítka, jeho puštění atd. a v závislosti na tom publikují eventy. Není jej možno instancovat.

`_Button` zároveň umožňuje určit klávesovou zkratku, která se bude chovat jako stisk levého tlačítka myši. Zkratku je možno určit pomocí pojmenovaných argumentů `shortcut_key` a `shortcut_keymod` při vytváření instance nebo při volání metody `set()`. Tyto pojmenované argumenty podporují i všechny widgety, které z `_Button` dědí. `shortcut_key` a `shortcut_keymod` určují po řadě hodnoty atributů `event.key` a `event.mod` pro eventy typu `pygame.KEYDOWN` a `pygame.KEYUP`. Klávesovou zkratku lze zatím přiřadit jen jednu ke každému widgetu a pouze pro levé tlačítko myši. Pro odstranění klávesové zkratky je nutno nastavit hodnotu atributů `shortcut_key` a `shortcut_keymod` na `None`.

`_Button` publikuje následující eventy:

`E_BUTTON_PRESSED`	Uživatel stiskl tlačítko myši nad tímto widgetem
`E_BUTTON_RELEASED`	Uživatel pustil tlačítko myši nad tímto widgetem
`E_BUTTON_BUMPED`	Uživatel nejdříve stiskl toto tlačítko a poté ho pustil, aniž by se mezitím kurzor dostal mimo tlačítko
`E_BUTTON_SLIDED`	Uživatel stiskl toto tlačítko a poté kurzorem pohnul mimo widget – tlačítko již není stisknuté, ale nepublikuje výše zmíněné eventy

E_BUTTON_MOUSEOVER	kurzor se dostal nad tento widget
E_BUTTON_MOUSEOUTSIDE	kurzor se pohnul mimo tento widget

Eventy E_BUTTON_PRESSED, E_BUTTON_RELEASED a E_BUTTON_BUMPED mají atributy button a pos. Atribut button nabývá hodnot BUTTON_LEFT, BUTTON_RIGHT, BUTTON_MIDDLE apod. a určuje stisknuté, případně uvolněné tlačítko myši. pos je poloha v okně, kde k tomu došlo. Pokud byla stisknuta či uvolněna klávesová zkratka, je button == BUTTON_LEFT a pos je uprostřed widgetu. Event E_BUTTON_SLIDED má atribut buttons, což je množina stisknutých tlačítek. V této množině může být i string 'shortcut', který udává, že v tom okamžiku byla stisknuta klávesová zkratka. Event E_MOUSEOVER má atribut pos, který určuje polohu kurzoru. Event E_MOUSEOUTSIDE nemá žádné speciální atributy.

Tato třída dědí ze třídy _Widget.

set(**kwargs) -> None

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z _Button, podporují všechny argumenty, které podporuje _Button.

shortcut_key	hlavní klávesa přiřazené klávesové zkratky
shortcut_keymod	modifikátory klávesové zkratky (např. Ctrl)

pygame_widgets.Button

standartní tlačítko s jednořádkovým textem

```
Button(master, topleft=(0, 0), size=(1, 1), **kwargs) -> Button instance
```

Argumenty pro instancování tlačítka mají stejný význam jako argumenty pro instancování Labelu.

Tato třída dědí ze tříd Label a _Button a implementuje pouze vzhled. Tlačítko má 3 typy vzhledu, mezi kterými se přepíná v závislosti na kurzoru. Také je možno vzhled přepnout pomocí pojmenovaného argumentu appearance metody set(). Hodnotou může být některý ze stringů 'normal', 'pressed' nebo 'mouseover'. Pokud je tlačítko zmáčknuté, je appearance == 'pressed'. Jinak je appearance == 'mouseover' v případě, že je kurzor nad tlačítkem, ve všech ostatních případech je appearance == 'normal'.

Pomocí pojmenovaných argumentů je možno nastavit vzhled pozadí a vzhled kurzoru pro jednotlivé módy vzhledu diskutované výše. Kurzor v módu 'normal' se řídí pojmenovaným argumentem cursor, který je zděděn z nadtřídy _Widget. Dále jsou však

přidány pojmenované argumenty `cursor_mouseover` a `cursor_pressed`, které řídí kurzor po řadě v módech `'mouseover'` a `'pressed'`. Pozadí v jednotlivých módech jsou určena po řadě pojmenovanými argumenty `bg_normal`, `bg_mouseover` a `bg_pressed`. Každý z nich se chová stejně jako pojmenovaný argument `background` metody `Label.set()`. Tento argument `background` je tlačítkem zděděn, je však silně doporučeno ho nepoužívat. V průběhu programu je totiž přepisován hodnotami jednotlivých módů.

`set(**kwargs) -> None`

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `Button`, podporují všechny argumenty, které podporuje `Button`.

Po každý nastavený atribut publikuje event `E_BUTTON_ATTR` (pro zděděné atributy) či `E_BUTTON_APPEARANCE` (pro vlastní atributy), který má atributy `name` (string obsahující jméno atributu), `new` (nově přiřazená hodnota) a `old` (původní hodnota; pro některé atributy rovna `None`).

`bg_normal`	`pozadí widgetu v módu 'normal'`
`bg_mouseover`	`pozadí widgetu v módu 'mouseover'`
`bg_pressed`	`pozadí widgetu v módu 'pressed'`
`cursor_mouseover`	`vzhled kurzoru v módu 'mouseover'`
`cursor_pressed`	`vzhled kurzoru v módu 'pressed'`
`appearance`	`nastavení aktuálního módu`

`pygame_widgets.Image`

widget stabilně zobrazující daný obrázek nebo barvu

```
Image(master, topleft=(0, 0), size=(1, 1), **kwargs) -> Image instance
```

Argumenty pro instancování obrázku mají stejný význam jako argumenty pro instancování `Labelu`. Obrázek dědí ze třídy `_Widget`.

`set(**kwargs) -> None`

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `Image`, podporují všechny argumenty, které podporuje `Image`.

Po každý nastavený atribut publikuje event `E_IMAGE_APPEARANCE`, který má atributy `name` (string obsahující jméno atributu), `new` (nově přiřazená hodnota) a `old` (původní hodnota; pro některé atributy rovna `None`).

image	callable, Surface nebo barva určující vzhled widgetu. Pro přesnější informace viz pojmenovaný argument background metody Label.set(). Jediný rozdíl je v tom, že callable dostane pouze argument self.

pygame_widgets.Holder

transparentní widget

```
Holder(master, topleft=(0, 0), size=(1, 1), **kwargs) -> Holder instance
```

Tento widget není dokončený. V budoucnu by měl sloužit k automatické organizaci widgetů na něm závislých, zatím ho lze použít pro "seskupení" widgetů, jejich hromadnému přesouvání atd. Také je to ideální widget pro vytváření vlastních widgetů.

Tato třída dědí ze třídy `_Widget`.

set(**kwargs) -> None

nastaví atributy widgetu

Všechny nastavitelné atributy každého widgetu se dají změnit pomocí této metody. Pro každý widget se liší seznam podporovaných názvů argumentů, přičemž všechny objekty, které dědí z `Holderu`, podporují všechny argumenty, které podporuje `Holder`.

Po každý nastavený atribut publikuje event `E_HOLDER_ATTR`, který má atributy `name` (string obsahující jméno atributu), `new` (nově přiřazená hodnota) a `old` (původní hodnota; pro některé atributy rovna `None`).

color	barva widgetu (uspořádaná trojice nebo čtveřice čísel, nebo instance pygame.Color)

create_row_layout() -> NotImplementedError

experimentální metoda

Jak bylo zmíněno výše, `Holder` zatím není plně dokončený. V budoucnu by měl podporovat vytváření tzv. layoutu (rozložení), s pomocí kterého by bylo možné závislé widgety např. automaticky přichytávat k hornímu okraji `Holderu`, k jiným závislým widgetům atd.

Tato metoda vyvolá `NotImplementedError`.

pygame_widgets.Entry

jednořádkové textové zadávací pole

```
Entry(master, topleft=(0, 0), size=(1, 1), **kwargs) -> Entry instance
```

Tento widget není dokončený, při pokusu o instancování vyvolá `NotImplementedError`. Widget by se měl chovat jako `Label`, který však mění text v závislosti na eventech typu `pygame.KEYDOWN` a `pygame.KEYUP`. Přesnější způsob ovládání, přizpůsobování a celkového fungování widgetu není zatím jistý, stejně jako podporované pojmenované argumenty.

`pygame_widgets.auxiliary.attributes.Attributes`

objekt umožňující ukládání atributů do jednotlivých widgetů bez rizika kolize s vestavěnými atributy používanými uvnitř knihovny

Každý widget (objekt dědící ze třídy `_Widget` nebo `Window`) má atribut `widget.attr`, ve kterém je uložena jedinečná instance objektu `Attributes`. Tento objekt implementuje řadu "systémových" metod (začínajících a končících dvěma podtržítky), díky kterým existuje několik způsobů, jak ukládat a následně číst atributy.

Prvním a pravděpodobně nejpoužitelnějším způsobem je přístup k atributům pomocí tečky:

```
widget.attr.the_answer = 42
print("The answer:", widget.attr.the_answer)  # "The answer: 42"
```

`Attributes` se však kromě toho chová i jako tzv. kontejner, přičemž oba přístupy lze kombinovat:

```
print(("The answer:", widget.attr["the_answer"])  # "The answer: 42"
widget.attr["the_answer"] = "nothing"
print("The answer:", widget.attr.the_answer)  # "The answer: nothing"
```

Je-li klíč typu string, je možné k hodnotě přistoupit pomocí tečky i pomocí hranatých závorek. Je-li však klíčem jakýkoli jiný objekt (samozřejmě musí být neměnný, viz dokumentaci objektu `dict`), není možné k hodnotě přistoupit přes tečku.

Chceme-li nastavit více atributů najednou, můžeme použít tento syntax:

```
widget.attr = value
```

Je-li `value` slovník nebo jiná instance třídy `Attributes`, všechny atributy budou zkopírovány. Je-li to iterovatelný objekt, bude rozbalen a jednotlivé hodnoty budou uloženy pod klíči `"attr0"`, `"attr1"` atd. Jinak bude objekt `value` uložen jako atribut pod klíčem `"attr0"`. Ve všech případech budou nejdříve vymazány všechny existující atributy. Potřebujeme-li k existujícím atributům přidat další atributy, můžeme použít „systémovou" metodu `__add__()`:

```
widget.attr += value
```

Opět, `value` může být slovník, instance třídy `Attributes` nebo iterovatelný objekt; ten bude také rozbalen obdobně jako v předchozím příkladě. Také je definováno odčítání:

```
widget.attr -= value
```

V tomto případě určují hodnoty v iterovatelném objektu `value` klíče, které mají být z `widget.attr` odebrány. Pro vymazání všech atributů je možné zavolat

```
del widget.attr
```

nebo použít metodu `widget.attr.clear()`.

`clear() -> None`

vymaže všechny uložené atributy

`toDict() -> dict`

vrátí slovník se všemi uloženými atributy

`pygame_widgets.auxiliary.handler.Handler`

objekt pro reprezentaci handleru

Pojmem „handler“ je většinou myšlena funkce, která je spuštěna při zachycení eventu. V mé knihovně je však možné, z důvodu větší variability a programátorské přívětivosti, nastavit i polohové a pojmenované argumenty, se kterými se má handler spustit (viz metodu `_Master.add_handler()`). Pro pohodlnější ukládání těchto informací byl implementován objekt `Handler`. Ten má tyto atributy:

```
func
args
kwargs
self_arg
event_arg
delay
```

Všechny tyto atributy jsou uložené argumenty metody `_Master.addHandler()`, v dokumentaci této metody je popsán jejich význam.

Objekt Handler implementuje systémovou metodu `__call__()`, která je spuštěna při přímém zavolání instance. V takovém případě dojde k zavolání `handler.func` s požadovanými argumenty. Povinnými atributy metody `__call__()` jsou `self` (reference na widget, jemuž handler patří) a `event` (reference na event, který je handlerem zpracováván).

`copy()`

zkopíruje handler

Vrátí novou instanci třídy `Handler`, která má stejné atributy jako `self`.

`pygame_widgets.cursors`

modul pro ovládání vzhledu kurzoru

Tento modul je wrapperem některých funkcí z modulů `pygame.mouse` a `pygame.cursors`. Umožňuje pohodlnější a snadnější práci než původní metody z pygame.

V pygame 1.9.6 je chyba v dokumentaci modulu `pygame.cursors`. Funkce `pygame.cursors.compile()` vrací 2 hodnoty, avšak funkce `pygame.mouse.set_cursor()` přijímá právě 4 argumenty. Proto nemůže tento kód uvedený v dokumentaci [20] fungovat:

```
cursor = pygame.cursors.compile(pygame.cursors.textmarker_strings)
pygame.mouse.set_cursor(*cursor)
```

Při spuštění bude vyvolán `TypeError`:

```
>>> import pygame
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
>>> cursor = pygame.cursors.compile(pygame.cursors.textmarker_strings)
>>> pygame.mouse.set_cursor(*cursor)
Traceback (most recent call last):
  File "<input>", line 2, in <module>
TypeError: function takes exactly 4 arguments (2 given)
```

Návratovými hodnotami funkce `pygame.cursors.compile()` jsou ve skutečnosti poslední 2 argumenty pro funkci `pygame.mouse.set_cursor()` (tedy `xormasks` a `andmasks`), zatímco `size` a `hotspot` musí zadat uživatel.

Modul `pygame_widgets.auxiliary.cursors` tento problém řeší tím, že poskytuje funkci `cursors.compile()`, která umožňuje zadání dodatečného argumentu `hotspot` a následně vrací všechny 4 požadované hodnoty.

Modul `pygame.cursors` obsahoval několik již zkompilovaných kurzorů a několik ve formě stringů. Všechny tyto kurzory jsou dostupné v modulu `pygame_widgets.auxiliary.cursors`, avšak ve zkompilované formě. Tento modul obsahuje také 2 nové kurzory. Toto je kompletní seznam dostupných kurzorů:

```
pygame_widgets.auxiliary.cursors.arrow
pygame_widgets.auxiliary.cursors.ball
pygame_widgets.auxiliary.cursors.diamond
pygame_widgets.auxiliary.cursors.broken_x
pygame_widgets.auxiliary.cursors.tri_left
pygame_widgets.auxiliary.cursors.tri_right
pygame_widgets.auxiliary.cursors.textmarker
pygame_widgets.auxiliary.cursors.thickarrow
pygame_widgets.auxiliary.cursors.sizer_x
```

```
pygame_widgets.auxiliary.cursors.sizer_y
pygame_widgets.auxiliary.cursors.sizer_xy
pygame_widgets.auxiliary.cursors.hand
pygame_widgets.auxiliary.cursors.none
```

Kurzor `none` může být argumentem funkce `cursors.set()`, ale nikoli funkce `pygame.mouse.set_cursor()`, na rozdíl od všech ostatních kurzorů.

`compile(strings, hotspot=(1, 1), black='X', white='.', xor='o') -> size, hotspot, data, mask`

vytvoří binární data kurzoru ze seznamu stringů

Tato funkce je v podstatě wrapper pro `pygame.cursors.compile()` s tím rozdílem, že vrací všechny potřebné argumenty.

`set(size, hotspot, xormasks, andmasks) -> None`

nastaví vzhled kurzoru

Tato funkce je wrapperem pro `pygame.mouse.set_cursor()`. Je-li však `size == None`, zneviditelní kurzor pomocí `pygame.mouse.set_visible()`. Dokáže tedy jako argument přijmout `pygame_widgets.auxiliary.cursors.none`.

`get() -> (size, hotspot, xormasks, andmasks)`

vrátí současné nastavení kurzoru

Tato funkce je wrapperem pro `pygame.mouse.get_cursor()`.

`load_xbm(cursorfile [, maskfile]) -> cursor_args`

načte data kurzoru ze souboru XBM

Tato funkce je wrapperem pro `pygame.cursors.load_xbm()`.

`pygame_widgets.constants`

modul obsahující velké množství konstant a pomocných funkcí

Tento modul je obdoba modulu `pygame.locals`. Ten je však napsán v jazyce C, proto nedokáží programovací prostředí jako např. PyCharm [21] rozpoznat názvy konstant a „našeptávat“. Programátor si poté musí všechny potřebné konstanty pamatovat nebo každou chvíli hledat v dokumentaci. Modul `constants` je však napsán v Pythonu, proto dokáže PyCharm přečíst všechny konstanty v něm definované.

Modul `constants` obsahuje všechny konstanty z `pygame.locals` díky příkazu

```
from pygame.locals import *
```

To však nevyřeší problém s „našeptáváním“. Proto jsem některé nejpoužívanější konstanty znovu definoval, čímž se staly viditelnými pro PyCharm. Mezi redefinované

konstanty patří vlajky (flags) pro `pygame.display.set_mode()`, typy eventů, klávesy, klávesové modifikátory, identifikátory tlačítek myši a vlajka `SRCALPHA`. Dále obsahuje modul `constants` referenci na objekty `Rect`, `color` a `Surface` a na databázi barev `pygame.colordict.THECOLORS`.

Knihovna pygame_widgets přidává další konstanty, kterými jsou ID eventů a konstanty zarovnání textu. Také jsou zde dostupné některé funkce, které mohou usnadnit používání pygame_widgets.

`Args(*args, **kwargs) -> args, kwargs`

vytvoří seznam a slovník z polohových a pojmenovaných argumentů

Prvotní účel této funkce je vytváření handlerů pomocí metody `_Master.add_handler()`. Ta totiž vyžaduje seznam a slovník jako argumenty `args` a `kwargs`, což je nepřehledné a zdlouhavé. Proto osobně preferuji následující syntax:

```
widget.add_handler(event_type, func, *Args(arg1, arg2, key=kwarg))
```

Obdobně lze funkci `Args` použít i pro metodu `_Master.remove_handler()`.

`button_wrapper(func, buttons=(BUTTON_LEFT,), self_arg=False, event_arg=False) -> handler_function`

wrapper, který zavolá handler pro *`_Button`* ***pouze při kliknutí danými tlačítky myši***

Návratovou hodnotu tohoto wrapperu je možné vložit jako argument `func` pro metodu `_Button.add_handler()` pro zpracovávání eventů `E_BUTTON_BUMPED` apod., které mají atribut `button`. Při zpracovávání takového eventu je zavolán wrapper, který zkontroluje, jestli bylo stisknuto (uvolněno atd.) některé z tlačítek přítomných v argumentu `buttons`. Také zkontroluje podpis eventu. Při splnění podmínek je zavolán argument `func` se všemi argumenty, které wrapper převzal.

Upozornění: Při vytváření handleru je nutné nechat argumenty `self_arg` a `event_arg` metody `add_handler()` na výchozí hodnotě, jelikož jsou použity k ověřování podmínek.

`button_bg(fill, edge, frame_thickness=1) -> func`

vrací funkci, která může být použita k vytváření pozadí labelu či tlačítka

Návratová funkce bude splňovat podmínky k tomu, aby mohla být použita jako pojmenovaný argument `background` metody `Label.set()`. Ve výsledném vzhledu bude label vyplněn barvou `fill` a bude mít obdélníkový rámeček šířky `frame_thickness` a barvy `edge`.

`frame(fill, edge, thickness=1) -> func`

vrací funkci vytvářející surface s rámečkem

Návratová funkce má povinný argument `size`, který společně s argumenty `fill`, `edge` a `frame_thickness` determinují obdélníkový rámeček.

Upozornění: Tato funkce bude pravděpodobně odstraněna.

7 ZÁVĚR

Knihovna pygame_widgets se stále vyvíjí a v nejbližší době plánuji několik zásadních změn, bude tedy potřeba průběžně aktualizovat i dokumentaci. Jak jsem však již zmiňoval v kapitole 6.3, bude nutné přeložit dokumentaci i do anglického jazyka. Z časových důvodů budu poté aktualizovat pouze anglickou dokumentaci. Obě jazykové verze budou dostupné v repozitáři https://github.com/Jajasek/pygame_widgets společně s aktuálním zdrojovým kódem knihovny.

8 POUŽITÁ LITERATURA

[1] Official Python documentation. [Online] [Citace: 13. březen 2020.] https://docs.python.org/3/.

[2] PyQt5 documentation. [Online] [Citace: 13. březen 2020.] https://www.riverbankcomputing.com/static/Docs/PyQt5/.

[3] Assembly - Basic Syntax. [Online] [Citace: 2. březen 2020.] https://www.tutorialspoint.com/assembly_programming/assembly_basic_syntax.htm.

[4] Strom. [Online] [Citace: 3. březen 2020.] https://cs.wikipedia.org/wiki/Strom_(graf).

[5] Turtle documentation. [Online] [Citace: 3. březen 2020.] https://docs.python.org/3/library/turtle.html.

[6] Tkinter documentation. [Online] [Citace: 3. březen 2020.] https://docs.python.org/3/library/tkinter.html.

[7] Tk documentation. [Online] [Citace: 3. březen 2020.] https://tkdocs.com/.

[8] Tkinter source - GitHub. [Online] [Citace: 3. březen 2020.] https://github.com/python/cpython/tree/3.8/Lib/tkinter.

[9] KSP - úloha 31-1-6. [Online] [Citace: 3. březen 2020.] http://ksp.mff.cuni.cz/tasks/31/tasks1.html#task6.

[10] Curses documentation. [Online] [Citace: 3. březen 2020.] https://docs.python.org/3/library/curses.html.

[11] Robot BREACH. [Online] [Citace: 3. březen 2020.] http://www.benderrobotics.com/breach.html.

[12] Pygame. [Online] [Citace: 3. březen 2020.] https://www.pygame.org/.

[13] SDL. [Online] [Citace: 3. březen 2020.] http://www.libsdl.org/.

[14] About pygame. [Online] [Citace: 3. březen 2020.] https://www.pygame.org/wiki/about.

[15] Pygame Intro - tutorial. [Online] [Citace: 3. březen 2020.] https://www.pygame.org/docs/tut/PygameIntro.html.

[16] Move It - tutorial. [Online] [Citace: 3. březen 2020.] https://www.pygame.org/docs/tut/MoveIt.html.

[17] SHINNERS, Pete. Python Pygame Introduction. [Online] [Citace: 3. březen 2020.] https://www.pygame.org/docs/tut/PygameIntro.html.

[18] Heroes of Might and Magic III. [Online] [Citace: 13. březen 2020.] https://cs.wikipedia.org/wiki/Heroes_of_Might_and_Magic_III:_Restoration_of_Erathia.

[19] Pampuch. [Online] [Citace: 13. březen 2020.] https://cs.wikipedia.org/wiki/Pampuch.

[20] Pygame cursors documentation. [Online] [Citace: 13. březen 2020.] https://www.pygame.org/docs/ref/cursors.html.

[21] PyCharm. [Online] [Citace: 13. březen 2020.] https://www.jetbrains.com/pycharm/.

[22] Pygame documentation. [Online] [Citace: 13. březen 2020.] https://www.pygame.org/docs/.

9 SEZNAM OBRÁZKŮ A TABULEK