Karel Šafr, Ph.D.

Pygame workshop

- pygame.org
- pygame.org/docs/ref

- Proč Pygame?
  - Jedná se o starší knihovnu, která je ale stále vyvíjená.
  - Je modulární, tj. lze část knihovny nahradit vlastní či pokročilejší.
  - Mnoho projektů se překvapivě navrhuje v Pygame, pro ostrý provoz pak se jednotlivé moduly nahrazují
    - např. GRM – grafic render machine – je jako vše ostatní separátní modul, tj. když je potřeba lepší grafiku, tak jej lze snadno nahradit jiným enginem -> OpenGL…
  - Obrovské množství dokumentace, kurzů, manuál, videí, knih…
  - Pygame je hodně populární díky modularitě, lze vytvářet i hry pro android (Pygame subset for android – PGS4A)
  - Lze kombinovat s matplotlibem, používá se také pro fyzikální a technické simulace a vizualizace.
  - Lze napojit i na kameru a základní computer vision v kombinaci s ML často poskytuje pohodlné prostředí pro monitoring.

- Co je Pygame?
  - Pygame je open-source knihovna pro vývoj her v jazyce Python.
  - Poskytuje sadu nástrojů a funkcí pro tvorbu 2D her.
  - Jednoduché použití a široká komunita uživatelů.
- Komponenty Pygame:
  - Grafika: Manipulace s obrázky, tvorba animací.
  - Zvuk: Přehrávání zvukových efektů a hudby.
  - Události: Zachytávání uživatelských vstupů jako klávesnice a myš.
  - Fyzika: Jednoduché simulace fyzikálních jevů.
  - Kolize: Detekce srážek mezi herními objekty.
- Přínosy používání Pygame:
  - Snadný vstup do vývoje her.
  - Přenositelnost mezi různými platformami.
  - Podpora mnoha herních prvků od grafiky po zvuk a interakce.

Možné aleternativy?

- Godot Engine:
  - Plně integrovaný herní engine s vlastním skriptovacím jazykem (GDScript).
  - Podpora 2D i 3D her.
  - Poskytuje mnoho vestavěných funkcí pro vývoj her.
- Pyglet:
  - Další populární knihovna pro vývoj her v Pythonu.
  - Zaměřuje se na grafiku, zvuk a události.
  - Nabízí vyšší úroveň abstrakce než Pygame, což může být výhodné pro určité projekty.
- Cocos2d:
  - Framework pro vývoj 2D her.
  - Poskytuje jednoduché a efektivní nástroje pro tvorbu her.
  - Podporuje animace, fyziku a další herní prvky.
- Panda3D:
  - Otevřený zdrojový herní engine.
  - Zaměřuje se na 3D vývoj her.
  - Poskytuje podporu pro Python, C++ a další jazyky.
- Arcade:
  - Jednoduchý a intuitivní herní framework.
  - Navržen pro vývoj 2D her.
  - Poskytuje podporu pro zpracování událostí, kolizí a animací.

Instalace + test

- Instalace se bude lišit podle OS
  - www.pygame.org/wiki/GettingStarted
  - V podstate na windows standardně **python –m pip install pygame**

- Nelze implementovat v jupyteru či v něčem obdobném co běží „mimo“ main.
  - Snahy o to jsou, ale obvykle to naráží na limity technologie.
- Nutné provést test zda vše ok (**bacha** na zvuk / uši / sluchátka !!!)
  - **python –m pygame.tests**

  nebo:
  - **py –m pygame.test**

  (podle toho jak máte python na pc…)
  - **python3 –m pygame.test**
  - **….**

```
Ran 1896 tests in 60.515s

OK
```

Vývoj her a obecně GUI v Pythonu

- Pro Python existuje poměrně velká škála knihoven pro vývoj her.
- Nejedná se jen o Pygame ale i o:
  - OpenGL / Vulkan
  - PyQT6
  - Pyglet (+3D support, cross platform; - Small comunity, pure python)
  - Kivy (+ cross platform, + Android / iOS extension; - malá komunita, Kv Design language)
  - Godot (+ má vlastní editor, build-in physics, lighweight; - ne dobře zdokumentovaný)
  - Ren‘Py(+ Cross platform, komunita, dobré pro 2D klikací hry; - cokoliv ne 2D težko, - horší pro „neklikací“,)

Proč hrají lidé hry?

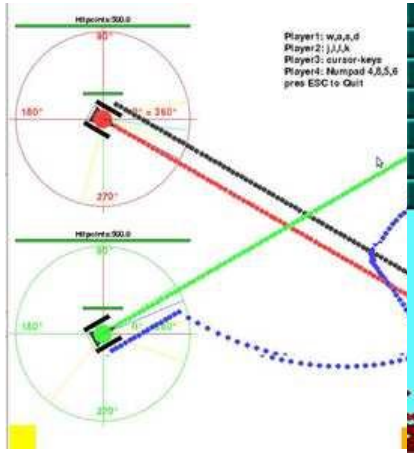
- Trochu filozofická otázka.
  - Obvykle se jedná o možnost si něco nacvičovat / procvičovat v neznámém prostředí, se kterým můžeme nějak interagovat.
  - 3 základní síly:
    - možnost manipulovat / měnit povahu jevů
      - velký pocit sebeuspokojení z pohledu toho, že hráč něco vytvoří -> hra je více playground
    - možnost vytvářet nové jevy
    - možnost ničit stávající jevy

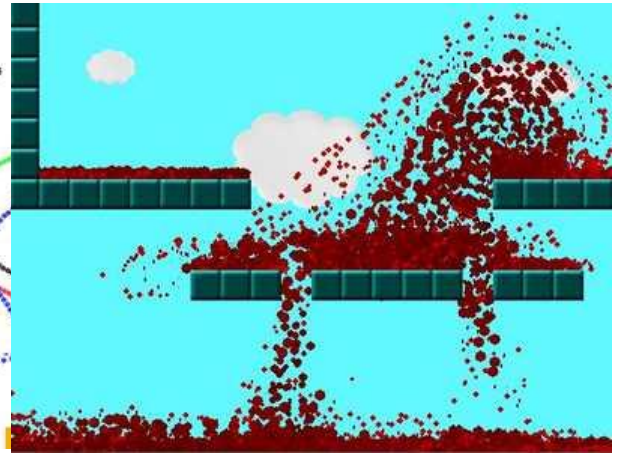
Pygame

- Pygame je modul pro vývoj her – herní knihovna
- Velmi dobře zpracovává interakci s uživatelem:
  - reakce na stisk kláves, myš, joystick… etc.
- Efektivní práce s multimédii:
  - animace
  - zvuk
  - obrázky…
- Lze v ní vytvářet i aplikace pro android !!!
- Pygame je vybudovaná nad SDL knihovnou !!!

Pygame – příklady her

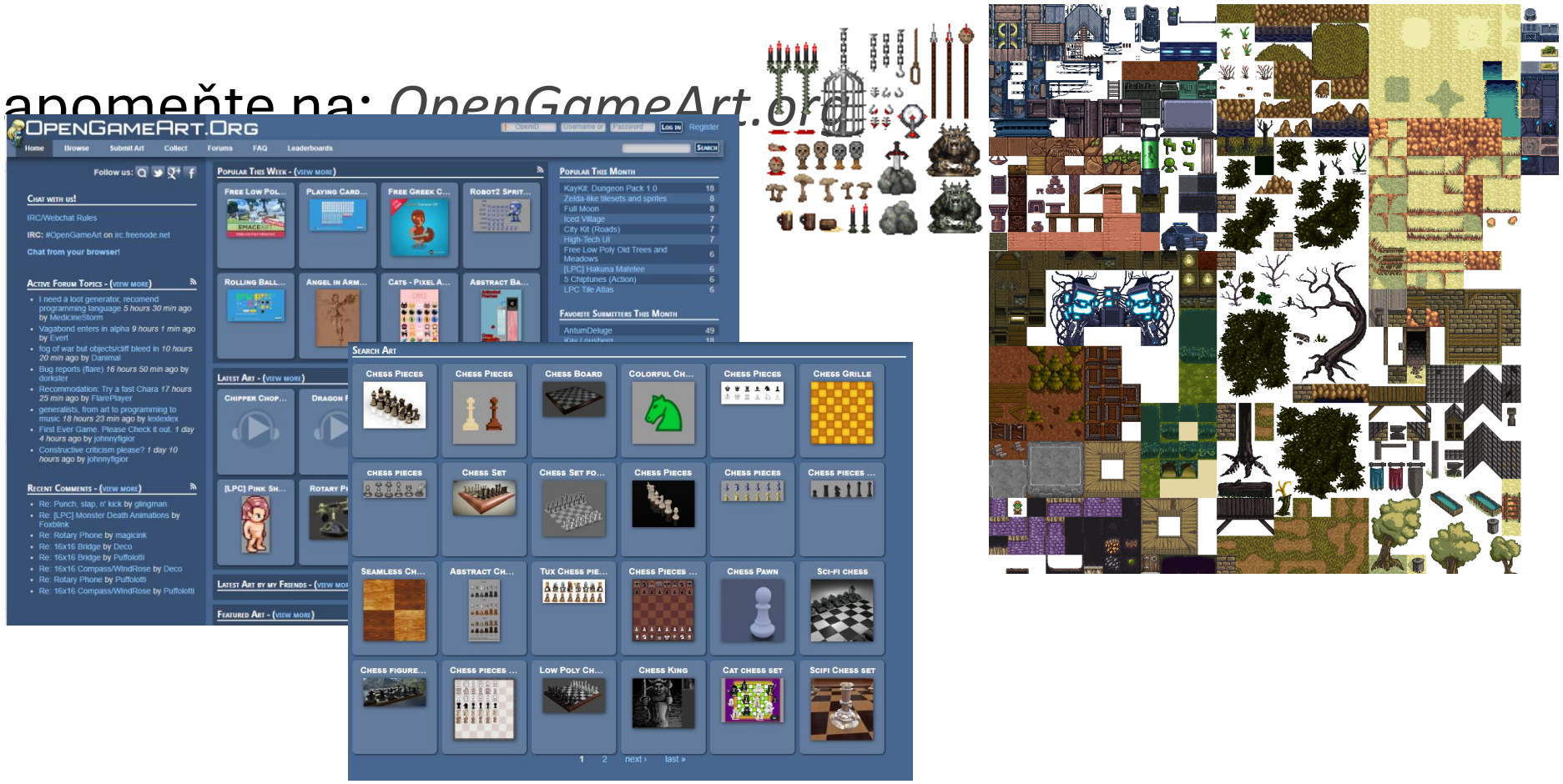
- Pří





OpenGameArt.org

- Nezapomeňte na: *OpenGameArt.org*



Pygame knihovny

- Pygame má mnoho dalších dílčích modulů. Je vhodné si je prozkoumat pomocí dokumentace!
  - Display
  - Draw
  - Event
  - Key
  - Time
  - Mouse
  - Transform
  - Joystic.....

Display

- Ovládání okna hry nebo obrazovky.
- **Funkce**:
  - set_mode(): nastaví velikost a vlastnosti okna
  - update(): aktualizuje celou plochu okna nebo jen určitou oblast
  - set_caption(): nastaví nadpis okna
  - flip(): aktualizuje celý displej pro 'double buffering'

Draw

- Nabízí funkce pro kreslení tvarů a linií na obrazovku.
- **Funkce**:
  - line(): kreslí linii mezi dvěma body
  - circle(): kreslí kruh na zadané pozici
  - rect(): kreslí obdélník na zadané pozici
  - polygon(): kreslí polygon z poskytnutého seznamu vrcholů

Event

- Správa událostí a fronty událostí.
- **Funkce**:
  - get(): získá seznam čekajících událostí
  - post(): může umístit novou událost do fronty
  - poll(): vyzvedne jednu událost z fronty událostí

Key

- Čte stav klávesnice nebo jednotlivých kláves.
- **Funkce**:
  - get_pressed(): vrátí stav všech kláves na klávesnici
  - set_repeat(): nastaví opakování kláves po jejich stisknutí
  - get_mods(): zjistí aktivní modifikační klávesy

Time

- Sleduje čas a ovládá snímkovou frekvenci hry.
- **Funkce**:
  - delay(): pozastaví program na určitý počet milisekund
  - tick(): sleduje a omezuje rychlost hry na FPS
  - get_ticks(): získá počet milisekund od startu hry

Mouse

- Sleduje pohyby a stisknutí tlačítek myši.
- **Funkce**:
  - get_pos(): získá aktuální pozici kurzoru myši
  - get_pressed(): kontroluje, zda jsou tlačítka myši stisknuta
  - set_visible(): skryje nebo zobrazí kurzor myši

transform

- Poskytuje operace pro změnu měřítka, otáčení a překlopení obrázků.
- **Funkce**:
  - scale(): změní měřítko obrázku na novou velikost
  - rotate(): otáčí zdrojový obrázek o určitý úhel
  - flip(): překlopí obrázek v horizontálním nebo vertikálním směru

Sprite

- Správa a grupování herních objektů.
- **Klíčové vlastnosti**:
  - **Sprite Class**: Základní stavební kámen pro vytváření herních objektů.
  - **Groups**: Třídy pro skupinové správu Sprite objektů, umožňuje efektivní rendering a aktualizaci.
  - **Sprite Collision**: Funkce pro detekci kolizí mezi Sprites.
- **Použití**:
  - Umožňuje snadné manipulace s mnoha objekty na základě jejich typů.
  - Podporuje dědičnost, takže můžete rozšířit základní Sprite a přidat vlastní funkce a atributy.

Ostatní moduly?

- Surface
  - V Pygame je Surface objekt reprezentující obrázkový blok pro ukládání a zobrazování grafického obsahu. Každé grafické zobrazení, které chcete vykreslit, musí být nejdříve načteno nebo vytvořeno na Surface objektu.
  - Obrázek v paměti = surface v pygame
- Rect
  - Obdélníkový objekt, který uchovává obdélníkové oblasti. Tato třída je velmi důležitá pro kolizní detekce a umístění a rozměry grafických prvků na obrazovce.
- Sound
  - Tento modul umožňuje pracovat se zvukovými efekty a hudbou ve vaší hře. Pomocí tohoto modulu můžete nahrávat, přehrávat a manipulovat se zvukovými soubory.
- Music
  - Pygame má speciální modul pro práci s hudbou na pozadí. Tento modul je optimalizován pro delší zvukové stopy a umožňuje přehrávání hudby ve smyčce nebo řízení přehrávání.
- Font
  - Pro zobrazení textu potřebujete tento modul, který umožňuje vytvářet fonty a vykreslovat text na Surface.

FPS

- FPS !!!
  - Frames per second
  - Jako u videa v kině / v pc – počet obrázků, které se zobrazí uživateli – animace, filmy aj. – vše jsou jen statické obrázky, které se rychle mění.
- Vysoký FPS může vést k hladkému a plynulému vizuálnímu zážitku.
- Nízký FPS může způsobit sekání a trhání obrazu, což ovlivňuje hratelnost a uživatelskou zkušenost.
- **clock**: Pygame poskytuje pygame.time.clock objekt pro ovládání FPS.
- **tick() metoda**: tick(framerate) reguluje hru tím, že omezuje běh smyčky na zadaný počet snímků za sekundu.

Jak funguje vývoj her a aplikací obe 

- V jakékoliv hře:

Herní vstup / event ->

aktualizace hry / parametrů objektů ->

vykreslení stávajícího stavu

- Co když má někdo rychlejší PC? Poběží mu hra rychleji?
- Co když pc nestíhá - FPS ???
- Dva přístupy: maximální FPS vs. FPS(t-1) + přepočet parametrů

Pygame – template – INIT/CONSTANTS

- Co musíme udělat?

1. Naimportovat potřebné knihovny (pygame, numpy, random…).
2. Deklarace konstant (width, height, fps…. fyzikální konstanty a jiné).
3. **pygame.init()**
    1. inicializace knihovny pygame
    2. může se nám dále hodit: **pygame.mixer.init()** – inicializace zvuku!
4. Nastavení obrazovky
    1. Např: **pygame.display.set_mode((width, height))**
        1. Obvykle se ukládá do nějaké proměnné !
    2. **pygame.display.set_caption(„jmeno hry“)**
5. Budeme používat přístup max FPS -> jednodušší na implementaci.
    1. Je nutné incicializovat „tikot hodin“, např:

    clock = pygame.time.Clock()
    - If change clock is something…..

Pygame – template – GAME CORE - EVENT

- 3 základní fáze - Event, Update, Render/Draw:
- **Event**:
  - Součástí eventů jsou pohyby myši, stisky kláves aj.
  - Nejstandardnějším eventem je zavření obrazovky s hrou!
    - Doposud to nic nedělá, když kliknete na X!
    - V rámci eventů zaznamenáme všechny provedené eventy od předchozího framu.
    - Pygame si pamatuje, jaké eventy byly provedené od posledního dotázání eventů!
    - Uložené v pygame.event.get() – vypsáním se zároveň vymaže! ☺

```
for event in pygame.event.get()
        if event.type == pygame.QUIT:
            running = False
```

    - Proč je toto důležité?
      - Jinak by se ověřovalo, zda v daný moment nedošlo ke stisku klávesy… což by asi nikdo na setiny vteřiny nestíhal přesně…

Pygame – template – GAME CORE - UPDATE

- 3 základní fáze - Event, Update, Render/Draw:
- **Update**:
  - Obvykle fáze, kde se dá způsobit to, že překročíte FPS.
  - Zde vznikají Lagy hry – přepočítávání způsobí, že dojde k překročení hodin.
  - clock.tick(FPS)

Pygame – template – GAME CORE – DRAW / RENDER

- 3 základní fáze- Event, Update, Render/Draw:
- **Draw**:
  - Překreslení objektu provedete pomocí například: screen.fill((0,0,0)).
  - Otočení překresleného snímku – cesta na další snímek:
    - pygame.display.flip()
    - DŮLEŽITÉ! Flip musí být proveden jako poslední věc cyklu běhu hry. Pokud to prohodíte, nebude to vidět :D

pygame.quit()

Inicializovat herní engine
Vytvořit herní okno
Nastavit název herního okna
Nastavit proměnnou running na true

While (running):
  Zpracovat události:
    Pokud je stisknuta klávesa ESC nebo uzavřené okno, nastaví proměnnou running na false
  Aktualizovat herní stav:
    Aktualizovat polohy objektů
    Kontrolovat kolize
    Zpracovat vstup hráče (klávesnice, myš)
    Aktualizovat stav hry (skóre, úrovně, atd.)
  Vykreslit herní stav:
    Vyčistit herní okno
    Vykreslit pozadí
    Vykreslit herní objekty
    Vykreslit uživatelské rozhraní
    Aktualizovat obrazovku (refresh)

Ukončit herní engine

01_Hello world

```
import pygame
pygame.init()
BLACK = (0,0,0)
PURPLE = (150, 10, 100)


screen = pygame.display.set_mode((800,600))
pygame.display.set_caption("Ahoj svete!")
screen.fill(PURPLE)
running = True
i = 1
# cyklus udrzujici okno v chodu
while running:
    # Event
    for event in pygame.event.get():
        print(event)
        if event.type == pygame.QUIT:
            running = False


    i+=1
    # Update
    myfont = pygame.font.SysFont("None",50)
    textik = myfont.render(f"Ahoj Svete!{i}",True, (250,80,100))
    # Render
    screen.fill((0,0,0))
    screen.blit(textik,(0+i,0+i))


    pygame.display.update()


pygame.quit()
```

Blok kódu

1. Import + inicializace

```
import pygame
pygame.init()
```

2. Konstanty barev

```
BLACK = (0,0,0)
PURPLE = (150, 10, 100)
```

3. Vytvoření okna

```
screen =pygame.display.set_mode((800,600))
pygame.display.set_caption("Ahoj svete!")
screen.fill(PURPLE)
```

4. Hlavní smyčka hry

```
while running:
```

- **EVENT**

```
for event in pygame.event.get(): ...
```

- **UPDATE**

```
i += 1
textik = myfont.render(...)
```

- **RENDER**

```
screen.fill(BLACK)
screen.blit(textik, (0+i, 0+i))
```

5. Zobrazení

```
pygame.display.update()
```

6. Ukončení

```
pygame.quit()
```

Co dělá & proč je důležitý

Načte knihovnu a aktivuje všechny moduly (video, zvuk…). Bez pygame.init() by se neotevřelo okno ani nešel font.

RGB trojice pro srozumitelný zápis barev; šetří magická čísla v kódu.

Otevře 800 × 600 px surface, nastaví titulek a počáteční výplň barvou pozadí.

Udržuje aplikaci „živou“. Běžně se jí říká **Game Loop** a dělí se na tři fáze:

Vytáhne vstupy (klik, klávesa, zavření okna) ze systémové fronty. Pokud přijde QUIT, nastaví running = False.

Logika hry – zde jen inkrement čítače a generování nového surface s textem. (V praxi se font typicky vytváří jednou a cache-uje.)

Vyčistí okno a vykreslí text o pár pixelů posunutý, čímž vzniká diagonální pohyb.

Po vykreslení všech elementů se surface „překlopí“ na obrazovku.

Uvolní prostředky Pygame a korektně zavře aplikaci.

02_ball

Základní rady & optimalizace kódu

- Nejpomalejší článek jakékoliv hry je překreslování objektů.
  - Nepřekreslujte to, co se nemění, pokud to jde = Double Buffering.
    - Back side – Rendering -> otočit -> Front Side – Display
    - Nepřekreslujte tedy každý objekt sám o sobě, ale vytvářejte FRAMY tak jako je to ve filmu !!!
    - Python dělá za vás – **pygame.display.flip()**
- **V dokumentaci k Pygame najdete stránku věnovanou se jen tomuto! Co používat a co ne aby jste dosáhli co největší rychlosti.**

Prvky Pygame:

- Sprites:
  - Objekty, které se realizují na obrazovce nějakým způsobem a nějakým způsobem interagují.
- Detekce kolize – interakce:
  - Všechny negrafické interakce objektů lze snadno vyřešit na nějaké úrovni kódu, ale v případě kolize řešíme na úrovni Pygame.

Sprites

- To, co vidíme na obrazovce – vizualizované objekty, které nějakým způsobem interagují s prostředím
- Sprites se obvykle skládají z dat a chování
- Nejčastěji pro ně řešíme:
  - Kartézské souřadnice
  - Obrázek
  - Velikost /škálování
  - Průchozí/neprůchozí – jak se chovat při kolizi
  - … a jestli stále jsou na obrazovce
- Pozor: **pygame.sprite.Sprite** -> každý sprite je subclass téhle třídy.

Základní grafické objekty

- Když potřebujete někam jen vykreslit základní grafické tvary, tak můžete použít:

pygame.draw.ellipse()

pygame.draw.circle()

pygame.draw.line()

pygame.draw.rect()

Update & Draw 2.0

- UPDATE:
  - Říká, že Sprites se mají nějak chovat, tedy „spouštějí“ jejich chování.
- DRAW:
  - Překresluje chování sprites.

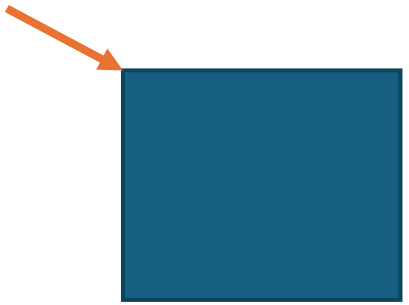
- Obvykle sprites držíme v kolekci, a pak je aktualizujeme a vykreslujeme najednou.

```
my_sprites = pygame.sprite.Group()
# do které je přidáme – mimo cyklus obvykle
#update:
my_sprites.update()
#draw
my_sprites.draw(screen)
```

- V případě vykreslení je nutné předat obrazovku, do které se to má vykreslit!

Sprites

- Pro každý sprite je nutné definovat širokou škálu metod a dat.
- Jedná se o předpřipravené sady jmen – tak jako například v třídách, např. __add__
- self.image – obrázek, který bude vykreslen
  - Nřp: **self.image = image.load(„letadlo.png“).convert()**
- self.rect – souřadnice a velikost – rozsah objektu (x, y, šířka, výška)
  - POZOR



- A pak sada metod:
- Colide… metody definující kolizi
-

Deklarace sprite

```
class kostka(pygame.sprite.Sprite)
        def __init__(self):
                pygame.sprite.Sprite.__init__(self)
                self.image = <nejaký způsob obrázku či objektu>
                self.rect = self.image.get_rect()
```

Obrázek můžeme:

```
        self.image = pygame.Surface((100,10))
        self.image = pygame.image.load(„ball.png“).convert()

        sefl.image.fill((100,160,30))
...
my_sprites.add(<sprite>)
```

Sprite.update

- Metoda update se volá pro každý sprite v cyklu - UPDATE part!
- Tj. vhodné definovat tuto metodu pro naše sprites – pokud chceme, aby něco dělal v čase!
- Např:

```
def update(self):
        self.rect.x += 5
        self.rect.y -= 1
```

03_kosticky - 1 varianta

Sprite - atributy

Atribut	Typ / výchozí hodnota	K čemu slouží	Praktický tip
image	pygame.Surface	Bitmapa, která se blituje na obrazovku.	Připravit při __init__() a pak jen měň referenci pro animaci (nepřetvářej Surface každým snímkem). (pygame.org)
rect	pygame.Rect	Ohraničující rámeček – určující pozici, velikost a kolize.	Pro plynulý pohyb ukládejte rychlost (vx, vy) a v update() volej rect.move_ip(vx, vy). (pygame.org)
mask	pygame.Mask *(volitelné)*	Per-pixel kolize pro sprite.collide_mask().	self.mask = pygame.mask.from_surface(self.image); po změně obrázku je potřeba masku znovu vygenerovat. (pygame.org)
dirty	int (1)	Říká LayeredDirty, jestli se sprite musí znovu překreslit (0 = ne, 1 = ano a vynuluj, 2 = překreslit vždy).	Nastavit na 2 u animovaných prvků, 0 u statických pozadí. (pygame.org)
visible	bool (1)	Když je 0, sprite se nevykreslí (ale může stále dostávat update).	Hodí se pro “blikání” po zásahu – jen vypnout a znovu zapnout. (pygame.org)
layer	int (RO)	Z-index používaný vrstvovými skupinami (LayeredUpdates, LayeredDirty).	Přidat sprite do skupiny s parametrem layer=n, nebo později group.change_layer(sprite, n). (pygame.org)
blendmode	int (0)	Hodnota předávaná jako special_flags do Surface.blit() – umožňuje např. aditivní míchání.	Pro efektní výbuchy nastavit blendmode = pygame.BLEND_ADD. (pygame.org)

Sprite – atribut rect

Postup	Co udělá	Vrátí	Mění původní Rect?	Kdy se hodí
rect.move(dx, dy)	Vypočítá nový obdélník posunutý o dx, dy.	**Nový** Rect	**Ne**	Chceteli krátkodobý výsledek (např. “kdybych sem šel, narazím?”) a původní souřadnice musí zůstat beze změny.
rect.move_ip(dx, dy)(„in place“)	Posune *ten samý* objekt.	None	**Ano**	Běžný pohyb sprite v update(), kdy se opravdu posouvá hráč/enemy.
Přímá změnar ect.x += dx rect.y += dy	Změní jednu nebo obě souřadnice.	N/A	**Ano**	Nejjednodušší zápis; ekvivalent k move_ip, jen bez tvorby dočasného obdélníku.

Sprite – atribut rect

- Proč existují dva způsoby (move vs move_ip)?
- **Efektivita paměti**
  - move() vytvoří kopii (nový objekt). Při pohybu stovek spritů v každém snímku by se tím zbytečně alokovalo a zahazovalo spoustu paměti.
- **Bezpečná “simulace nanečisto”**
  - Když potřebujete zjistit, kam by se sprite dostal, ale nechceš mu změnit souřadnice (např. před kontrolou kolize), sáhnete po move().

- Chci pohyb? -> move_ip nebo: s.rect.x += vx; s.rect.y += vy

Sprite - metody

Metoda	Signatura	K čemu slouží	Tip z praxe
update	update(*args, **kwargs)	Hook pro logiku sprite – volá ho Group.update().	Předejte dt = clock.get_time()/1000 a díky tomu bude pohyb nezávislý na FPS. (pygame.org)
add	add(*groups)	Přidá sprite do jedné či více skupin.	Můžete volat dynamicky (např. při spawnu střely) – není drahé. (pygame.org)
remove	remove(*groups)	Odebere sprite ze zadaných skupin.	Když pracujete se skupinami podle rolí (např. enemies, bullets), uvolní to rovnou všechny reference. (pygame.org)
kill	kill()	Okamžitě vyřadí sprite ze **všech** skupin.	Ideální pro “smrt” objektu; po kill() můžeš instanci znovu oživit pomocí add(). (pygame.org)
alive	alive() -> bool	Vrací True, pokud sprite patří aspoň do jedné skupiny.	Rychlý filtr v update smyčce: if not enemy.alive(): continue. (pygame.org)
groups	groups() -> list[Group]	Vrátí seznam skupin, v nichž sprite právě je.	Užitečné při ladění: print(player.groups()). (pygame.org)

EVENTS 2.0

- Za event považujeme cokoliv, co se stane – mimo rámec hry! A chceme, aby ta hra na to reagovala! - Nejde o něco, co nastane v rámci hry samo!
- -> kliknutí myší, zmáčknutí klávesnice
- Chceme, abychom mohli ovládat naši kostku pomocí šipek! ☺
- Pomůže nám kolekce (list) zmáčknutých kláves:

Keys= pygame.key.get_pressed()

- A následně je to nutné ověřit, zda je zmáčknutá daná klávesnice, na kterou to má reagovat:

```
if Keys[pygame.K_LEFT]:
        ....
```

Máme klávesy jako:

pygame.K_RIGHT, pygame.K_UP, pygame.K_DOWN, pygame.K_a, pygame.K_b…

03_kostka2 a 3

UPDATE 3.0 – spridecollide()

- Zjišťování toho, zda se jeden objekt dotkl druhého, je poměrně náročné. Musí se:
  - Zjistit plochu rect pro všechny objekty a zjistit překryvy – poměrně dost náročné i na naprogramování.
- Pygame to umožňuje zjistit poměrně jednoduše pomocí příkazu:

Kolize= pygame.sprite.spritecollide(sprite, group2, False/True)

- Třetí výraz slouží k tomu, zda se má odstranit sprite z group2, který byl zasazen spritem. Zde se jedná o porovnání 1 sprite s kolekcí sprites!

If Kolize:

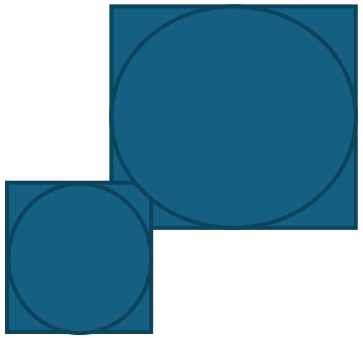
running = FALSE

#... nebo cokoliv jiného ☺

- **pygame.sprite.groupcollide(group1,group2, False/True, False/True)**
- Toto ověřujeme po výrazu my_sprites.update()!
- Group(s) – jedná se o kolekci sprites, které jsou jak v my_sprites, tak i v group!
  - Musí se jednat o stejný objekt!!!
- Kolize dvou spritů: **<sprite>.rect.colliderect(<sprite2>)**

UPDATE 3.0 – kolize

- Někdy objekty, které do sebe narazí, nejsou hranaté.
- Dojde ke kolizi z titulu toho, že do sebe narazili jednotlivé rect
- Ale ne již samotné objekty.
- Toto lze řešit několika způsoby:
  - list rect
  - použití jiného objektu než obdélník.



Nejčastější řešení je využití kruhu:

poloměr:

self.radius = 50

jak je velký radius to zjistíme kdyžtak pomocí:

pygame.draw.circle (self.image, (0,255,0),self.rect.center,self.radius)

a upravit specifikaci typu kolize:

pygame.sprite.spritecollide(nas_sprite,group1,False/True,
pygame.sprite.collide_circle)

Pohybující se kostka 3 a kolize

Grafika 2

- Obecně:
  - Natahujeme grafiku najednou – před spuštěním samotného cyklu – lagy.
  - Vše konvertujte!
- Pozadí:
  - Již jsme si ukazovali, že nejjednodušší způsob, jak vyplnit pozadí, je nějakou barvou pomocí:

screen.fill((0,0,0))

  - Obrázek na pozadí lze nejlépe vložit jako rect objekt. BLITneme ho nad tohle pozadí!
  - Mimo cyklus:

Background = pygame.image.load(„nasepozadi.png“).convert()

  - V cyklu:

screen.fill((0,0,0)
screen.blit(background,backround_rect)

04_hrac_v_plose

Kamera a hráč v ploše k 04

- Logika “player-centric” kamery – po každém kroku se spočítá camera_x / camera_y = player_pos − half_screen, takže obraz vždy posouvá svět tak, aby hráč zůstal opticky uprostřed.
- Omezení kamery na hranice mapy – dvojité max()/min() zajistí, že pokud by se střed hráče dostal blíž k okraji, kamera zůstane zarovnaná na hranu a neukáže “prázdno” mimo mapu.
- Oddělení světových a obrazovkových souřadnic – hráč i dlaždice se drží v globálních (mapových) hodnotách; při vykreslení se od nich odečte camera_*, čímž se převedou do lokálních (pixelů na monitoru).
- Render dlaždic ve dvou cyklech s posunem – smyčky for x / for y generují grid po 100 px; souřadnice se při volání draw.rect posunou −camera_x / −camera_y, aby se čtverce správně “scrollovaly”.
- Sub-pixelová rychlost hráče – player_speed = 1.5 je float, takže pohyb je plynulejší než čistě po integerových krocích; při vykreslení se stejně konvertuje implicitně na int.
- Neustálá re-centrace vs. samotné souřadnice – ačkoli kamera se každým framem mění, skutečné player_x, player_y pořád rostou v rozsahu 0 … MAP_SIZE; díky tomu lze později doplnit kolize či minimapu bez přepočítávání.
- Smyčka jen vykresluje to, co “vidí monitor” – všechny dlaždice mimo viewport se sice procházejí v cyklu, ale jejich rect může vyjít zcela mimo (screen.blit by se dal ještě filtrovat pro výkon).
- Mapu lze horizontálně/vertikálně rozšířit bez změny kódu kamery – stačí upravit MAP_WIDTH / HEIGHT; vzorec pro kameru i okrajové clamping zůstanou univerzálně platné.

05_Template

Pygame template

- Oddělení čisté logiky od Pygame-kódu – barvy, konstanty rozlišení / FPS i “ne-pygame” funkce jsou nahoře, takže se dají importovat a testovat bez spuštění okna.
- Jednorázová inicializace knihovny – pygame.init() (+ volitelné pygame.mixer.init()) spustí všechny interní moduly dřív, než budete načítat grafiku či zvuk.
- Přehledné místo pro načtení assetů – sekce “Grafika!” a “Definice spritu” slouží ke čtení obrázků a vytvoření tříd; díky komentáři studenti vědí, kam svůj kód vkládat.
- Konfigurace okna & titulku – set_mode((WIDTH, HEIGHT)) a display.set_caption() zabalena do jediné části; změna rozlišení se provede právě tady.
- Centrální FPS-hodiny – objekt clock + clock.tick(FPS) stabilizuje počet snímků a zároveň vrací časový krok, pokud byste chtěli delta-time pohyb.
- Skupinový management spritů – my_sprites = pygame.sprite.Group() umožňuje hromadné update() a draw(), místo abyste každý objekt volal zvlášť.
- Hlavní smyčka se třemi fázemi – komentovaná struktura Event → Update → Render ukazuje standardní pattern Pygame her a usnadňuje studentům orientaci.
- Kontrolované ukončení – test na pygame.QUIT nastaví running = False, takže se cyklus ukončí a pygame.quit() korektně uvolní prostředky.
- Jednoduchá rozšiřitelnost – díky čistým placeholderům (grafika, sprite-třídy) stačí doplnit vlastní logiku bez zásahu do already-working skeletonu.

06_CAR

06_CAR

- Modul-nezávislé načítání souborů – všechna PNG se hledají s os.path.dirname(__file__), takže skript běží správně i po přesunu do jiné složky nebo na jiný OS bez tvrdých cest.
- Car jako plnohodnotný fyzikální objekt – drží vektor rychlosti, natočení, omezení maximální rychlosti a parametry akcelerace/brzdy; pracuje s pygame.
- Vector2, takže výpočty směru a délky jsou kompaktní.
- Friction založené na driftu – rozdíl úhlu mezi směrem auta a směrem pohybu zvyšuje koeficient tření, takže smyk intuitivně ztrácí energii rychleji než jízda rovně.
- Per-frame rotace spritu – pygame.transform.rotozoom() otáčí původní obrázek podle aktuálního self.angle; díky convert_alpha() zůstává průhledné pozadí a auto se točí bez artefaktů.
- Mantinely i „vnitřní ostrov“ z generovaných Wall-spritu – okraje obrazovky plus obdélník uprostřed se vloží do jediné skupiny walls, což usnadňuje kolize a vykreslování.
- Kolize řešené jednoduchým odrazem – po srážce se rychlost vektoru násobí −1.5 v obou osách, takže auto odskočí a zpomalí zároveň; výsledek je rychlý a bez složité fyziky.
- Jeden sdílený background – textura se načte a hned přeměřítkuje na velikost okna; blit se provede před zdmi i auty, takže celý level má jednotné „podloží“ bez dalších draw-volání.

Problém s rotací obrázku

- Rotace je interpolace – nikoli čistý přenos pixel : pixel. -> Každé otočení musí přepočítat barvu mnoha bodů z nových plovoucích souřadnic; výsledkem je malý „šum“ a někdy i rozmáznutí.
- Když otočíte už jednou otočený obrázek, interpolujete dvakrát. Tyto malé chyby se sčítají – hrany se postupně rozmazávají a barvy šednou.
- Bounding-box roste, protože diagonální pixely vyčnívají dál než původní pravoúhlý obrys. Při opakované rotaci se každý další výsledek bere jako vstup a znovu se „obkreslí“ větším obdélníkem.
- Po několika snímcích auto „tloustne“ – rect nabere 1 – 2 px navíc, takže může narážet na kolizní zdi dříve a vizuálně vypadá roztahaně.
- Řetězová rotace je i dražší na CPU/GPU, protože operace běží víc krát, místo aby se počítala jen jednou z původních dat.
- Správná cesta: vždy rotovat původní, bezztrátovou kopii (original_image). Interpolace se provede jen jednou a kvalita i rozměr zůstanou stabilní.
- Rect po otočení nastavíme podle středu starého rectu, abychom zachovali polohu auta na mapě, i když se jeho obdélník ve skutečnosti „obtočí“.

Chybně a správně

CHYBNĚ – kumulativní rozmazání a zvětšování – car2

- self.image = pygame.transform.rotozoom(self.image, -delta_angle, 1)

SPRÁVNĚ – pokaždé z čistého zdroje

- self.image = pygame.transform.rotozoom(self.original_image, -self.angle, 1)
- self.rect = self.image.get_rect(center=self.rect.center)

Jednorázová animace

- Často se nám bude hodit informace o čase který uběhl!
- pygame.time.get_ticks()- v milisekundách
  - prozradí čas, který proběhl od posledního updatu – to vám umožní vytvářet animace, které nebudou podléhat různému FPS!
- Pak stačí přidat do update části sprite objektu příslušné obrázky animace.
- Obvykle se vytvoří speciální sprite s animací, která se zavolá na potřebném místě!
  - Animace je jen střídání obrázků s nějakým časováním ☺
  - Po dokončení animace je vhodné samozřejmě sprite odebrat!

Grafika 2.1

- Sprites
  - Grafiku stačí vložit místo našich čtverečků do self.image.
  - Může se nám stát, že budeme potřebovat přeškálovat obrázek, pak:

  self.image= pygame.transform.scale(<nas brazek>,(20,80))
  - Pokud budete něco škálovat, tak se hodí zachovat poměry stran!
  - Černého pozadí se zbavíme tak, že nastavíme color key!

  self.image.set_colorkey((0,0,0))
- pygame.transform.flip(<obrazek>,True/False, True/False)
  - True False určují otočení obrázku – flip vůči jaké ose.

Obrázek používaný právě jedním spritem

- I tak načíst jen jednou
- nejlépe v `__init__()` toho sprite, případně v loaderu scény. `convert_alpha()` zůstává důležitý.
- Vyhnete se „lazy“ načtení až při prvním renderu (což by způsobilo hitch)
- Když sprite zanikne, Python povolí garbage-collect i surface → RAM se uvolní.

```
class Boss(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("boss.png").convert_alpha()
        self.rect = self.image.get_rect(midbottom=(400, 500))
```

Jeden zdroj + úpravy (re-color, otáčení)

- Pořád sdílet originál v cache.
- Pro unikátní varianty použit `copy()` → `pygame.transform.` → uložte výsledek do jiného cache slotu.
- **Úpravy mění pixely**; kdyby jste je dělili, všechny stromy by zčervenaly naráz.
- Kopie se vytvoří jen jednou, ne každým snímkem.

```
if "tree_red" not in cache:
    red = cache["tree"].copy()
    pygame.PixelArray(red).replace((34,139,34), (220,20,60))
    cache["tree_red"] = red
```

Spritesheet (více rámů v jednom PNG)

- Načtěte sheet do cache -> `convert_alpha()`.
- Připravte **sub-surfaces** (nebo `blit` do nových surface) a uložte do listu pro animaci.
- **Sub-surface sdílí paměť s originálem** ⇒ jeden velký obrázek v RAM, rychlý slicing.

```
sheet = cache["hero_sheet"]
frames = [sheet.subsurface(pygame.Rect(x*32, 0, 32, 32)) for x in range(4)]
```

Škálování / rotace

- Nikdy** nevolat `transform.scale` nebo `rotate` každý frame
- Přepočítej jednorázově na cílové rozlišení (nebo pár předem daných úhlů) a uložte
- Transformace jsou CPU-náročné; předpočítaná sada surface → plynulé FPS

Singleton

```
_res_cache = {}

def load_image(name, colorkey=None):
    if name not in _res_cache:
        img = pygame.image.load(f"assets/{name}").convert_alpha()
        if colorkey is not None:
            img.set_colorkey(colorkey)
        _res_cache[name] = img
    return _res_cache[name]
```

Takový jednoduchý “singleton” garantuje, že se každý soubor čte z disku právě jednou. |

Shrnutí principů

1. Načtěte a `convert_alpha()` hned po startu** – žádná zbytečná I/O pauza během hry.

2. Sdílej surface mezi sprity, pokud pixely zůstávají stejné.

3. Kopírujate jen při úpravě (tint, rotace, maska).

4. Používejte globální nebo scénový cache/loader místo „importu obrázku“ v každé třídě zvlášť.

- Dodržením těchto čtyř bodů minimalizujete jak paměť, tak náhodné záseky a máte jistotu, že hra poběží plynule i s mnoha stovkami instancí na scéně.

08_IMAGE - kód

0x_SIM: Simulace - kovid

- Co je simulace?
  - Program, který napodobuje reálný systém (epidemii, fyziku, ekonomiku) → cílem není „zvítězit“, ale porozumět chování modelu.
- Proč je užitečná?
  - bezpečné testování „co-kdyby“ scénářů bez nákladů v reálném světě
  - rychlá iterace parametrů (n = 100 → 10 000 jedinců, změna pravděpodobností…)
- vizualizace dat (grafy, heatmapy) usnadní prezentaci výsledků ne-programátorům
- Proč Pygame?
  - jednoduchý 2D engine → rychlé vykreslení agentů, bez OpenGL/GUI knihoven
  - jednotný event loop; můžeme v real-time přepínat parametry, pauzovat, zobrazit graf
  - Surface jako plátno i pro matplotlib screenshoty → graf a simulace v jednom okně
- Jak to řeší ukázkový kód:
  - Model – třída Individual s atributy infekce, imunity, rychlosti, stavu alive/dead.
  - Logika – update() posun + stochastické přenosy, žádný vstup hráče.
  - Vizualizace – pygame.sprite.Group.draw() + tři grafické funkce pro průběžné statistiky.
  - Oddělení dat ↔ renderu – stejný back-end by šel nahradit třeba CSV exportem, ale Pygame poskytuje okamžitou „živou“ zpětnou vazbu.

0x_SIM: Simulace - kovid

Zvuk

- Pygame má komplexní přístup ke zvuku.
- Ideální je Pygame.mixer = ogg / wav files
- Je nutné inicializovat knihovnu s zvukem:

pygame.mixer.init()

- Pak stačí zvuk natáhnout do objektu:

my_sound = pygame.mixer.Sound(<file.wav>)

- A spustit:

my_sound.play()

09_FIGHTER + animace a zvuk

Generované pozadí z Tiles

- Inicializace Pygame – pygame.init() spustí všechny potřebné moduly (video, vstup, atd.)
- Nastavení okna – set_mode((320, 320)) vytvoří 320 × 320 px Surface, kam se bude vykreslovat.
- Načtení sprite sheetu – pygame.image.load("DungeonCrawl...png") přečte celý PNG s dlaždicemi do paměti.
- Definice konstanty tile_size – hodnota 32 určuje, že každá dlaždice má 32 px na šířku i výšku.

Generované pozadí z Tiles

- Funkce get_tile(x, y) – vrací sub-surface vyříznutý z sprite-sheetu; souřadnice se násobí tile_size, takže (1, 13) znamená pixel (32, 416).
- Výběr konkrétních dlaždic – volání get_tile() uloží tři Surface: <grass_tile>, <water_tile>, <desert_tile>.
- Mapovací matice tile_map – 10×10 seznam čísel (0 = tráva, 1 = voda, 2 = poušť) popisuje, co se má vykreslit v každé buňce.
- Funkce draw_map() – dvojité for (enumerate) prochází řádky (y) i sloupce (x) matice.
- Přepočet na pixely – souřadnice buňky se násobí tile_size, aby se tile blitoval na správné místo na obrazovce.
- Výběr Surface podle čísla – if/elif větví vybere jednu ze tří předpřipravených dlaždic.

Generované pozadí z Tiles

- screen.blit() – kopíruje vybranou dlaždici na cílový Surface (okno).
- Vyplnění pozadí – před každým snímkem screen.fill((0, 0, 0)) vynuluje obrazovku černou barvou.
- Hlavní smyčka while running – běží dokud proměnná running zůstává True.
- Zpracování událostí – pygame.event.get() vytáhne frontu událostí (klik, stisk klávesy, zavření okna…).
- Reakce na QUIT – když uživatel zavře okno, nastaví se running = False a smyčka skončí.

Generované pozadí z Tiles

- Volání draw_map() – každý frame překreslí celou mřížku dlaždic podle aktuální matice.
- Přepnutí bufferu – pygame.display.flip() zobrazí právě vykreslený snímek na monitoru (double-buffering).
- Konec programu – po ukončení smyčky pygame.quit() korektně uvolní zdroje Pygame.
- Statická vs. dynamická scéna – protože mapu nic nemění, každý frame se znovu kreslí totéž; na malém rozlišení to nevadí, ale u větších map by se vyplatilo předrenderovat.
- Rozšiřitelnost – díky funkci get_tile() lze snadno přidat další typy dlaždic jen doplněním indexu a další podmínky v draw_map() (nebo lepší – pomocí slovníku).

Proč je to dobré?

- Uvidíte, jak efektivně pracovat se spritesheetem – naučíte se „vystřihnout“ dlaždice jedním příkazem subsurface a pochopíte, proč je sdílení jednoho obrázku paměťově výhodnější než desítky samostatných PNG.
- Pochopíte převod z mřížky na pixely – ukážeme, jak se indexy [řádek][sloupec] v matici promění na souřadnice x × TILE_SIZE, y × TILE_SIZE, což je základ každého tile-based level-designu.
- Odnesete si princip oddělení dat a vykreslování – jasně uvidíte, proč je vhodné držet mapu (tile_map) jako datovou strukturu a samotné kreslení v samostatné funkci draw_map().
- Procvičíte si úplnou herní smyčku – seznámíte se s cyklem událostí → aktualizace → render → flip a uvidíte, jak správně reagovat na událost QUIT.
- Získáte přehled o prvních krocích optimalizace – ukážeme, kde má smysl použít convert_alpha(), před-renderovat statické pozadí nebo nahradit řetězec podmínek slovníkem, abyste připravili kód na rozsáhlejší mapy.

10_Tiles

Isometric tiles

- Izometrická dlaždice = diamantová „kostička“ – obrázek v poměru 2 : 1 (typicky 64 × 32 px), který působí jako šikmo položená čtvercová plocha.
- Předstírá 3D prostor bez 3D grafiky – stačí 2D obrázky, ale hráč vnímá hloubku, výšku objektů i překrývání budov.
- Jednoduchá logika mřížky – tvůrce hry pracuje s obyčejnou tabulkou [y][x]; konverzní vzorec převádí souřadnice na pixely.
- Plynulé scrollování a vrstvení – dlaždice lze skládat do nekonečných map, přidat parallax pozadí nebo vrstvy (podlaha, objekty, stíny).
- Úspora výkonu – oproti plnému 3D nejsou potřeba složité modely ani světla; stačí pár set textur a rychlý 2D blit.
- Snadné animace a výměna skinů – každou dlaždici lze nahradit jinou bitmapou (např. podlaha → láva) bez změny kódu fyziky či AI.
- Přehlednost pro hráče – izometrický pohled ukazuje současně „nadhled“ i boční stěny; hráč lépe vnímá rozmístění jednotek i bariér.
- Estetika klasických strategií a RPG – styl evokuje hry jako Diablo II, SimCity 2000 či Age of Empires a dodává retro-/pixel-artovou atmosféru.

Isometric tiles

- Inicializujete Pygame a otevřete okno 800 × 600 px. To vytvoří hlavní Surface, na který se budou blitovat všechny dlaždice a hráč.
- Definujete rozměry izometrické dlaždice 64 × 32 px. Poměr 2 : 1 dává klasický „diamantový“ tvar vycházející z projekce 30 °/30 °.
- Ukládáte velikost mřížky 10 × 10 polí. Každé pole existuje jen jako dvojice souřadnic (x, y); grafika se dopočítá při vykreslení.
- Funkce grid_to_iso(x, y) převádí mřížkové souřadnice na obrazovku. Vzorec (x − y)*½ TILE_W, (x + y)*½ TILE_H posouvá diamanty do správných pixelů a zarovnává mřížku na střed.
- draw_grid() projde všechna pole a kreslí zelené diamanty. Bodovou čtveřici polygonu počítá z vrcholu, poloviny a paty dlaždice, takže obrys přesně sedí.

Isometric tiles

- Izometrické dlaždice se kreslí shora dolů (většinou podle x + y). Díky tomu objekty „vzadu“ překreslí ti vepředu a perspektiva působí správně.
- Hráč drží souřadnice v mřížce (player_x, player_y). Žádné pixely se neukládají; poloha se vždy převádí na izometrické pixely před vykreslením.
- draw_player() zobrazí hráče jako červený kruh uprostřed dlaždice. Přičítá polovinu výšky dlaždice, aby kruh ležel přesně „na zemi“.
- Pohyb řešíte čtyřmi šipkami, jedna dlaždice za stisk. Kontrola hran mřížky brání výběhu mimo mapu; pygame.time.delay(100) vkládá krátký interval mezi kroky.
- Hlavní smyčka běží na 60 FPS a zpracuje události i vstup. Po každém snímku se okno promaže na černou a znovu překreslí celá mřížka + hráč.
- Isometrické vykreslování zjednoduší level-design: místo ručního počítání pixelů si vystačíte s logickou maticí a konverzí přes grid_to_iso.
- Princip je univerzální: stejné vzorce a vrstvení použijete pro statické pozadí, texturované dlaždice i vyšší 3D objekty (domy, stromy) – stačí změnit obrázek místo polygonu.

11_Isometric tiles

12_Isometric tiles

13_miny

13_miny

Zadání práce

Povinné „minimální“ požadavky:

- Na začátku úvodní obrazovka a po pár sekundách přepnutí do menu
- Menu + tlačíko hrát
- Několik typů spritů
- Použití spritesheetů a tedy tiles
- Animace
- Zpracování vstupů od uživatele (nějaké reakce na klávesy)
- Zpracování kolize a reakce spritů na kolizi
- Počítání bodů za nějakou interakci (např. životy, zabití, gól, prolítlá brána…) a jejich vykreslení aktuálního stavu

Volitelné

- Zvuk

Knihovny instalované nad základní sadu knihoven pythonu: Jen Pygame a Numpy nic víc.

Jaké hry nědělat

Pong – dva pádla, odrážející se míček, scoreboard.

Breakout / Arkanoid – pádlo + zdi z cihel, které postupně mizí.

Tetris – padající tetromina, rotace, řádkové mazání a zvyšující se rychlost.

Snake – had, který se prodlužuje po snědení jablka, kolize se stěnou/vlastním tělem.

Asteroids – trojúhelníková loďka, střelba do rozpadávajících se asteroidů, setrvačnost.

Space Invaders – řady nepřátel postupující dolů, kryty před zásahy, high-score.

Flappy Bird klon – nekonečný side-scroll, průlet mezi trubicemi, fyzika gravitace.

Platformer à la „Mario“ – běh, skok, kolizní dlaždice, nepřátelé, power-upy.

Top-down Shooter – pohled shora, WASD + míření myší, nekonečné vlny.

Tower Defense – mřížka cest, stavění věží, vlny nepřátel se zdravím.

Pac-Man – labyrint, sběr puntíků, pronásledování duchů s AI stavem (scatter/chase).

2048 – logická hra se sléváním čísel na čtyřech směrech.

Minesweeper – odhalování políček, označování min, flood-fill algoritmus.

Sudoku / Nonogram – vygenerovaná logická mřížka, validace řešení.

Šachy a obdobné hry – tahová logika figur, zvýraznění možných tahů, základní AI (minimax).

Klikací „Cookie Clicker“ – idle mechanika, počítadlo, upgrade tlačítka.

Memory (pexeso) – otáčení dvojic karet, sledování pokusů a časovače.

Jednoduchý závodní „top-down“ okruh – kolize s okrajem dráhy, měření kola.

komplexní již ano ☺

Simulátor životního cyklu částic – generování a pohyb „jisker“ s časovým útlumem.

Tedy

Originální hru chci!

Co odevzdáte?

- HRU (sadu vlastní kodu)
- Dokumentaci ke hře(pdf)
- Prezetnaci hry (prezentace v pdf):10-20 slidů kde bude popsaná herní mechanika, vysvětlení řešení v kódu
  - Na úvodním slidu bude jméno skupiny + jméno i každého člověka a co dělal (kod-jaká část zhruba aj)
- + Sady obrázků ze hry – pro případ kdyby se nepodařilo mi hru pustit/ či to blblo at vím jak má hra vypadat a co hledat opravit
- Neodevzdání dané části je penalizované příslušným počtem bodů.

Kde získat zdarma grafiku?

- Můžete použít AI generovanou grafiku

Nebo:

- https://opengameart.org/
- https://kenney.nl/
- https://itch.io/game-assets/free
- https://craftpix.net/freebies/
- https://lospec.com/
- https://www.spriters-resource.com/
- https://ambientcg.com/
- https://assetstore.unity.com/top-assets/top-free