Programowanie gier w PyGame

Łukasz Milewski

Uniwersytet Wrocławski

April 20, 2011, Wrocław

Target

Dla kogo jest ten wykład?

- Początkujący programiści
- Osoby chcę tworzyć gry dla zabawy
- Osoby oczekujące 'szybkich' efektów
- Chętni do uczenia się nowych rzeczy

Dla kogo to nie jest wykład?

- Nastawieni na komercyjny gamedev
- Zaawansowani programiści
- Osoby chcące tworzyć gry na konsole

OOP, DOD, CBP i inne literki

Gra amatorska to nie komercyjne oprogramowanie

Czego nie warto robić

- OOP, DOD, CBP, TDD i inne literki
- Clean code
- Pisanie własnego frameworka, enginu, czegokolwiek
- Uogólnianie

Co warto robić

- Pisać grę
- Dodawać kolejne featury
- Stosować KISS
- Maksimum efektów przy minimalnym nakładzie pracy

System kontroli wersji

github

- https://github.com/
- http://help.github.com/create-a-repo/

git

- git clone
- git push origin master - za pierwszym razem
- git commit -a -m 'tutaj opis zmian'
- git push
- gitk

System kontroli wersji

git a paraca grupowa

- git remote add NAZWA ADRES
- git fetch NAZWA
- git merge NAZWA/master (integrator)
- git rebase NAZWA/master (collaborator)

Motywacja

- Programowanie z kimś jeszcze (jak wybrać taką osobę?)
- Konkursy (np. www.pyweek.org, compo)

Dwie najskuteczniejsze metody nauczenia się progrmowania gier

Pisz gry

Praktyka czyni mistrza. Co więcej - każda, nawet najprostsza, skończona gra daje sporą dawkę motywacji do tworzenia kolejnych. Uczenie się z tutoriali jest nieefektywne.

Czytaj kod

Na www.pyweek.org oraz www.pygame.org jest bardzo wiele przykładowych gier. Warto wybierać te, które wygrywały poprzednie edycje pyweeka i zobaczyć jak są zrobione. Kod zazwyczaj nie jest piękny, obiektowy czy zgodny z inną ideologią. Za to działa, jest skończony, rozwiązuje konkretne problemy i ma więcej featurów niż pozostałe gry w danej edycji.

Dlaczego warto wybrać Python

Zalety

- Pygame
- Bardzo łatwy do nauczenia się
- REPL (toplevel)
- Można zrobić kompletną grę w tydzień
- Dużo bibliotek
- Bogata biblioteka standardowa (np. moduł random)

Wady Pythona

Wady

- Wydajność
- Wydajność
- Wydajność

Python - składnia

liczby, booleans, słowniki, krotki, if, listy, for, range

Zobacz data_types.py

slicing, list comprehensions

Zobacz lists.py

funkcje, klasy, metody

Zobacz fun_class.py

Python - debugging

Jak wyszukiwać błędy?

- użyj debuggera [restart, p, c, b] (prezentacja)
- print debugging (moduł pprint)
- bisekcja

Python - moduły

sys

- import sys
- sys.argv (sound = not "–nosound" in sys.argv)
- sys.argv[0]

os

- import os
- base_path = os.path.abspath(os.path.dirname(sys.argv[0]))
- os.path.* (join, abspath, dirname, isfile)

Struktura katalogów

Płaska struktura katalogów

1. font/
2. gfx/
3. music/
4. sounds/
5. src/

Pliki źródłowe

Warto trzymać wszystkie pliki źródłowe - np. pliki .xcf, .psd itp.

Ważne pliki

Źródła

1. const.py
2. config.py
3. sprite.py
4. resources.py
5. main.py

Dobrze jest zrobić te obiekty jako zmienne globalne (słowo kluczowe global)

Podstawy

Klasa Game

1. def init(self, screen)
2. def update(self, dt)
3. def display(self, screen)
4. def process_event(self, event)
5. self.is_finished

Kontekst

W konstruktorze tworzymy obiekty gry oraz sprite'y. Do obiektów dobrze jest przekazać obiekt Game (self)

Podstawy

Pętla główna

main_loop.py

TBA

```
speed = self.current\_speed()
px, py = self.position
dx, dy = self.direction
px, py = px + dx * dt * speed, py + dy * dt * speed
```

Ładowanie zasobów

Zasoby

- pygame.image.load("obrazek.png").convert_alpha()
- sound = pygame.mixer.Sound("dzwiek.ogg")
- music - jest streaming. nie trzeba niczego ładować
- pygame.font.Font("font.ttf", size)
- load_animation.py

Wykorzystanie zasobów

Zasoby

- screen.blit(img, position)
- sound.play(loop)
- pygame.mixer.music.load(name),
  pygame.mixer.music.play(repeat)
- text_img = font.render(text, 1, color)

Przetwarzanie wejścia/wyjścia

- polling
- zdarzenia events.py

Kolizje

Kolizje z główną postacią

```
for enemy in self.enemies:
if aabb_collision(self.ferris.aabb(), enemy.aabb()):
self.die()
return
```

kolizje - odległość

```
def distance(pos1, pos2):
dx = pos1[0] - pos2[0]
dy = pos1[1] - pos2[1]
return math.sqrt(dx*dx + dy*dy)

if distance(player, register) < 5:
player.pick(register)
```

Kolizje

kolizje AABB

def aabb_collision((minx1, miny1, maxx1, maxy1), (minx2, miny2, maxx2, maxy2)): xcollision = (minx1 <= minx2 and minx2 <= maxx1) or (minx2 <= minx1 and minx1 <= maxx2) ycollision = (miny1 <= miny2 and miny2 <= maxy1) or (miny2 <= miny1 and miny1 <= maxy2) return xcollision and ycollision

Pixel perfect

Bardzo dokładne kolizje, jednak w Pythonie trudno jest zrealizować z powodu dużego kosztu obliczeniowego. Dlatego odradzam ich stosowanie.

Kolizje

Gdzie wykrywać kolizje?

- W klasie Player
- W klasie Game

pygame.surface

- blit
- convert, convert_alpha
- copy
- set_at
- get_at

pygame.draw

- rect
- polygon
- circle
- ellipse
- arc
- line
- lines
- aaline
- aalines

pygame.Rect

- copy
- contains
- collidepoint
- colliderect
- collidelist
- collidelistall
- collidedict
- collidedictall

pygame.transform

- flip
- scale
- rotate
- laplacian
- average_surfaces
- average_color

inne

- pygame.display.set_caption('Title')
- pygame.mouse.set_visible(True/False)
- pygame.sprite
- ...

Przydatne adresy

Linki

- http://www.python.org/doc/
- www.pygame.org
- http://www.pygame.org/docs/ref/
- www.pyweek.org
- http://github.com
- https://github.com/lmmilewski/pygame_basics