Multimedia-Programmierung
Übung 5

Ludwig-Maximilians-Universität München
Sommersemester 2013
Today

• Animations with Pygame

Literature: W. McGugan, Beginning Game Development with Python and Pygame, Apress 2007
Objects on the Screen don’t actually move

- Basically, only the colours of pixels are changed
- Everytime something changes, the whole screen is repainted
- Framerate defines the appearance of the animation (the higher, the better)
- Possible framerate depends on the hardware (e.g. hertz of the monitor)
import pygame
from pygame.locals import *
from sys import exit

player_image = 'head.jpg'
pygame.init()

screen = pygame.display.set_mode((640, 280), 0, 32)
pygame.display.set_caption("Animate X!")
mouse_cursor = pygame.image.load(player_image).convert_alpha()

x = 0 - mouse_cursor.get_width()
y = 10

while True:
    event = pygame.event.get()
    if event.type == QUIT:
        exit()

    screen.fill((255,255,255))
    if x > screen.get_width():
        x = 0 - mouse_cursor.get_width()
    screen.blit(mouse_cursor, (x, y))
    x+=10
    pygame.display.update()
Timing and Framerate

- Problem: The previous example creates an animation that runs in different speed depending on the power of the cpu
- Solution: *time-based animations*
  - `pygame.time.Clock()` provides an appropriate tool for time-based animations
  - `Clock.tick()` returns the time that passed since its last call

```python
clock = pygame.time.Clock()
clock.tick()
```
Moving an object time-based

```python
clock = pygame.time.Clock()
speed = 300.0
x = 0 - mouse_cursor.get_width()
y = 10
while True:
    time_passed = clock.tick() / 1000.0
    moved_distance = time_passed * speed
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((255,255,255))
    if x > screen.get_width():
        x = 0 - mouse_cursor.get_width()
    screen.blit(mouse_cursor, (x, y))
x+=moved_distance
    pygame.display.update()
```

- **speed** in pixels per second
- **time passed since last tick()** in seconds
- **distance moved since last call**
- move the sprite the calculated distance
Diagonal Movement
or: Vectors, yeah!

- Moving a sprite to a specific coordinate requires movement on the x- and y-axis
- Best achieved using vectors
- E.g. a vector of (10,30) means move 10 pixels on the x- and 30 on the y-axis
- Store vectors as tuples, lists or create a class

\[(10,30)\]  \[\text{vector} (30,-20)\]  \[(40,10)\]
Vectors I

• Example class

```python
class Vector(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "vector (%s,%s)"%(self.x, self.y)

@classmethod
def vector_from_points(cls,from_p,to_p):
    return cls(to_p[0]-from_p[0],to_p[1]-from_p[1])

vector1 = Vector(10.0,20.0)
print vector1

print Vector.vector_from_points((10,10), (30,10))
```

Use:

```python
vector1 = Vector(10.0,20.0)
print vector1

print Vector.vector_from_points((10,10), (30,10))
```

Output:

```
vector (10.0,20.0)
vector (20,0)
```
Vectors II

- Vector magnitude

```python
import math
class Vector(object):
    ...

    def get_magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)
```

A (10, 30)  Vektor (30, -20)  B (40, 10)

\[ \sqrt{30^2 + (-20)^2} \]
Vectors 3

• Normalizing a vector

```python
class Vector(object):
    ...

    def normalize(self):
        magnitude = self.get_magnitude()
        self.x /= magnitude
        self.y /= magnitude
```

(x, y) = (10, 30)

(40, 10)

y = -0.55

x = 0.83
import pygame
from pygame.locals import *
from sys import exit
import math

class Vector(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __str__(self):
        return "(%s, %s)"%(self.x, self.y)

    def get_magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)

    def normalize(self):
        magnitude = self.get_magnitude()
        self.x /= magnitude
        self.y /= magnitude

@classmethod
def vector_from_points(cls, from_p, to_p):
    return cls(to_p[0]-from_p[0], to_p[1]-from_p[1])
Diagonal movement using vectors 2

...  start and end positions
mpos = (0.0, 0.0)  
destination = (500, 430)
player_image = 'head.jpg'
pygame.init()

screen = pygame.display.set_mode((640, 640), 0, 32)
pygame.display.set_caption("Animate X!")

mouse_cursor = pygame.image.load(player_image).convert_alpha()
clock = pygame.time.Clock()

speed = 300.0 # pixels per second
heading = Vector.vector_from_points(mpos, destination)
heading.normalize()  calculate the vector and normalize it

...
Diagonal movement using vectors 3

... within the event loop ...  

while True:  
  for event in pygame.event.get():  
    if event.type == QUIT:  
      exit()  
    screen.fill((255,255,255))  
  
  time_passed = clock.tick() / 1000.0  
  moved_distance = time_passed * speed  
  
  screen.blit(mouse_cursor,mpos)  
  mpos= (mpos[0]+heading.x * moved_distance,mpos[1] + heading.y * moved_distance)  
  pygame.display.update()  

... the distance is calculated as usual  

new position is based on the normalized vector
Rotating Surfaces

- Use `pygame.transform.rotate(surface,angle)` to rotate a surface (counterclockwise)
- Returns a new Surface Object
- **Attention**: the new Surface can have different width and height than the original

\[
\text{rotated\_surface} = \text{pygame.transform.rotate(}\text{old\_surface},90)\]

![Diagram showing rotation of a surface](image)