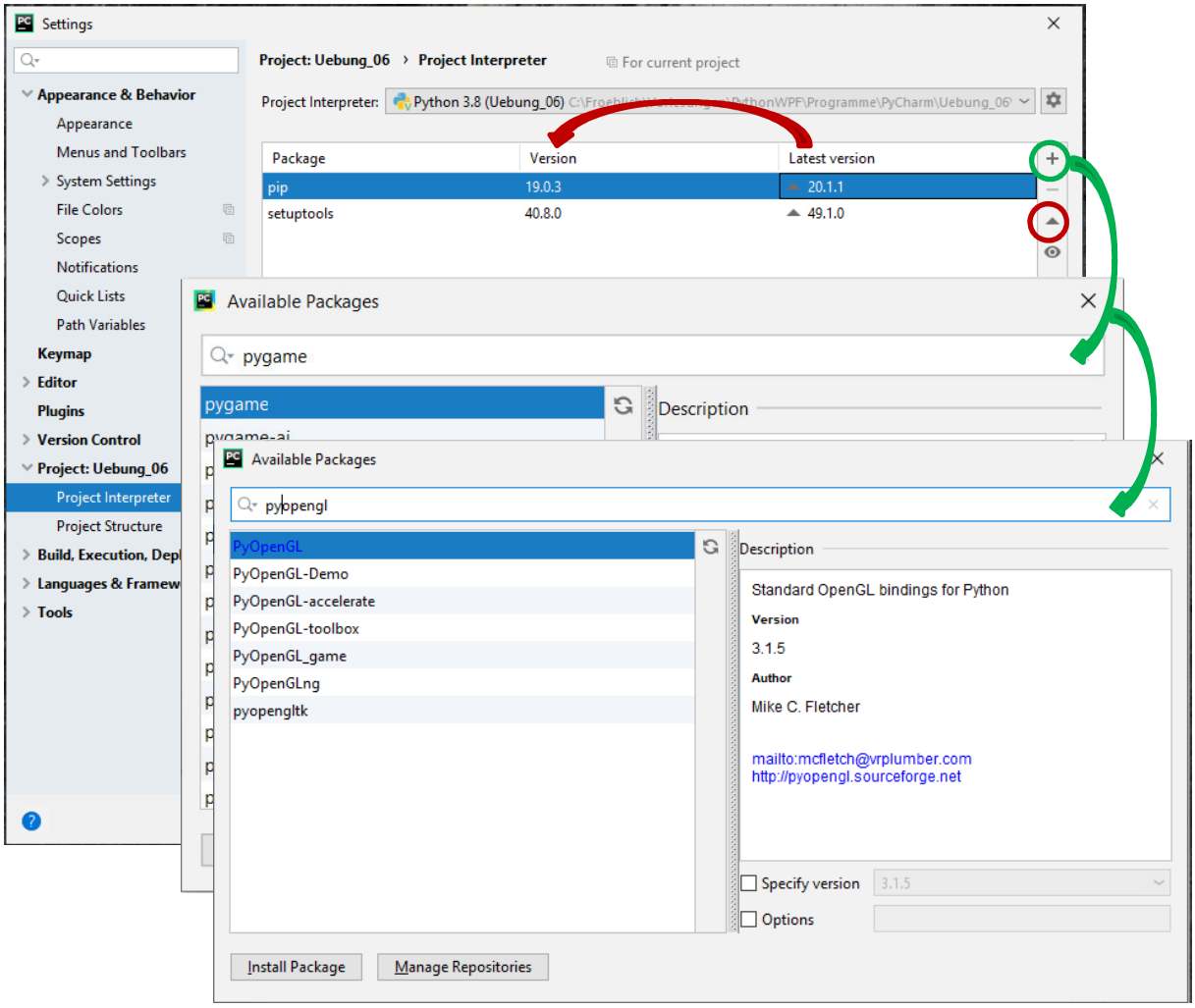
Übungsblatt 5 (Graphische Dimensionen)

Aufgabe 1: Alles ein bisschen square!?

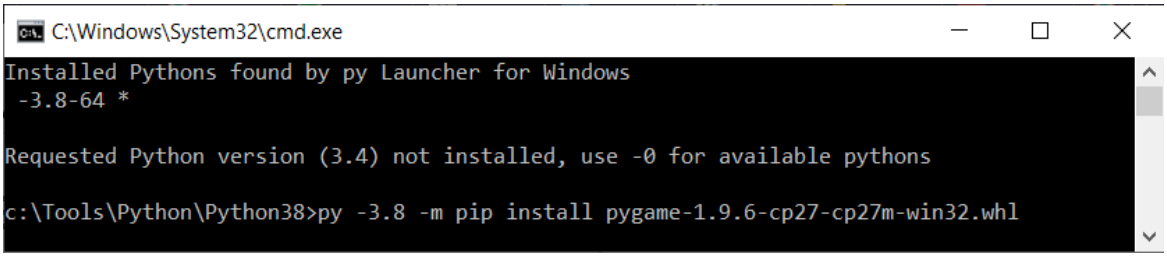
Zunächst erstellen Sie ein neues Projekt zum Übungsblatt 5 in Ihrer Entwicklungsumgebung.

Prüfen Sie, unter File->Settings… ob pip in einer aktuellen Version vorliegt! Falls das nicht der Fall ist muss pip mit dem kleinen Pfeil rechts aktualisiert werden (oder alternativ in Ihrer venv-Umgebung durch das Kommando: `python –m pip install -- upgrade pip`)

Importieren Sie danach pygame und pyopengl in Ihr Projekt.



Bei Problemen können Sie auch versuchen pygame und pyopengl manuell in Ihre venv zu integrieren:



a) Erzeugen Sie ein neues Python-Skript „MainWindow“ und kopieren Sie folgende Zeilen in das Skript:

```
import pygame                               # to use pygame keyboard and mouse control
from pygame.locals import *                 # to use constants from pygame directly
from OpenGL.GL import *                     # to use features from OpenGL

class MainWindow:                           # Class and constructor for main Window
   def __init__(self,windowHeight = 480, windowWidth = 640, name = "Main Window"):
       pygame.init()                                    # invoke initial settings
       pygame.display.set_caption(name)                 # set window title
       self.__windowSize = (windowWidth, windowHeight)# set size (default VGA)
       self.__videoFlags = OPENGL | DOUBLEBUF           # use OpenGl and double Buffer

       # create window with given size and specified video options
       self.__window = pygame.display.set_mode(self.__windowSize, self.__videoFlags)

    def close(self)
        pygame.quit()                           # quit pygame to close window
```

b) Erzeugen Sie ein weiteres Skript für das Hauptprogramm (z.B. Aufgabe_01.py):
- importieren Sie das Skript zur Klasse MainWindow
- erzeugen Sie eine Fenster mit der Default-Größe und Namen
- lassen Sie Ihr Programm warten durch: `input("Press the Any-Key")`
- schließen Sie das Fenster

Das Programm sollte jetzt ein Fenster öffnen mit dem von Ihnen gewählten Namen! Leider können Sie mit diesem Fenster noch nicht viel anfangen! Noch nicht einmal ordentlich schließen können Sie das Fenster, da wir noch keine Routine zum Behandeln der Mouse-Events haben!

c) Erzeugen Sie ein Fenster mit Default Größe aber selbst gewähltem Namen, dass Sie nach einer Tasteneingabe wieder schließen können

d) Erzeugen Sie danach ein Fester beliebiger Größe und selbst gewähltem Namen, dass Sie nach einer Tasteneingabe wieder schließen können

Aufgabe 2: Der Event-Händler

a) Informieren Sie sich wie der einfachste Event-Handler in einer Endlosschleife unter pygame aussieht:

   https://www.python-lernen.de/grundgeruest-fuer-pygame.htm

   Ab Schritt 4 wird diese erklärt! Setzen Sie einen einfachen Event Handler direkt unter dem bisherigen Code des Skriptes MainWindow.py um.

   Implementieren Sie dazu die Methode `mainloop(self)` in der Klasse `MainWindow` aus Aufgabe 1.

b) Fangen Sie die Position der Mouse innerhalb der Endlosschleife ab und geben Sie diese mit print() auf der Konsole aus.

Hinweis zu weiteren Quellen:

https://www.spieleprogrammierer.de/wiki/Pygame-Tutorial
https://riptutorial.com/pygame/example/18046/event-loop
http://www.pygame-doku.laymaxx.de/ref/event.html

Nun haben Sie ein Fenster, das ordnungsgemäß läuft und per Maus bzw. ESC-Taste geschlossen werden kann!

Zudem haben Sie Grundwissen darüber, wie und wo wir das Zeichnen einer Graphik in dieser Endlosschleife einbinden können.

c) Schreiben Sie ein Python-Skript (z.B. Aufgabe_02.py), dass ein Fenster bestimmter Größe öffnet und danach die „Endlosschleife“ des Event-Handlers startet.
   Nach Rückkehr aus der „Endlosschleife“ können Sie nochmals eine Tastenabfrage einbauen oder einfach das Skript durch `quit()` beenden.

d) Wechseln Sie die Hintergrundfarbe zufällig jede Sekunde unter Verwendung des OpenGL Buffers:

```
glClearColor(R, G, B, alpha)          # define background color and alpha of window
glClear(GL_COLOR_BUFFER_BIT)  # clear OpenGL-Buffer and set background color
```

   Hinweise:
   - die Werte R,G,B und alpha liegen zwischen 0 und 1 und definieren die RGB-Farbanteile sowie die Transparenz
   - random.random() liefert eine Zufallszahl im Bereich von 0 bis 1

Zusatzaufgabe:

Ermöglichen Sie, dass die main-loop mit 60 fps läuft (also die Tastatur- und Maus-Abfrage schnell läuft), sich die Hintergrundfarbe jedoch trotzdem langsam ändert mit einem Wechsel pro Sekunde.

Verwenden Sie dazu die Möglichkeit Timer in Python zu definieren: `pygame.time.set_timer()`

Aufgabe 3: OpenGL hat Ecken und Kanten!

OpenGL ist eine Graphik-Library, mit der es möglich ist perspektivische Graphiken zu definieren und in einem Fenster darzustellen. Dazu muss zunächst diese Graphik in Form von Eck-Punkten (engl. vertices) und Kanten (engl. edges) definiert werden.

a) Erzeugen Sie ein weiteres Skript "Objects2D" und kopieren Sie folgendes Skript hinein:

```
from OpenGL.GL import *

class Objects2D:                                    # base class for 2D-objects
    def __init__(self, xPos, yPos, color):
        self.xPos = xPos                            # store center x-position
        self.yPos = yPos                            # store center y-position
        self.R, self.G, self.B = color              # unfold color tuple
        self.alpha = 1                              # set transparency to none
        self._edges = list()                        # create list for edges
        self._vertices = list()                     # create list for vertices

    def update(self):                               # update drawing of 2D-object
        glBegin(GL_LINES)                           # draw lines, only
        for edge in self._edges:                    # iterate all edges
            for vertex in edge:                     # iterate all vertices to draw
                glColor4f(self.R, self.G, self.B, self.alpha) # red,green,blue,alpha
                glVertex3fv(self._vertices[vertex])
        glEnd()                                     # End of definition

class Square(Objects2D):                            # subclass to define 2D-squares
    def __init__(self, xPos = 0, yPos = 0, color = (1, 1, 1), size = 1):
        Objects2D.__init__(self, xPos, yPos, color) # call base class constructor
        self.__size = size                          # store size (for later use!?)
        dx = dy = size / 2                          # to set square to the center
        self._vertices = [(xPos - dx, yPos - dy, 0), (xPos + dx, yPos - dy, 0),
                          (xPos + dx, yPos + dy, 0), (xPos - dx, yPos + dy, 0)]
        self._edges = [(0, 1), (1, 2), (2, 3), (3, 0)]
```

b) Erweitern Sie Ihr Hauptprogramm aus Aufgabe 2 (z.B. als Aufgabe_03.py) so, dass Sie nach der Erzeugung des pygame-Fensters eine leere Liste für Objekte anlegen, ein erstes Quadrat mit square() erzeugen und an die Liste mit append() anhängen.

c) Noch können Sie nichts sehen! Zunächst muss die Endlosschleife von pygame erweitert werden, da wir noch gar keine Graphik darstellen dazu erweitern Sie die Endlosschleife der Methode von MainWindow und übergben die Objektliste übergeben (evtl. benennen Sie das Skript neu zu MainWindow2D):

```
def mainLoop(self, objectList):
    running = True                          # flag to stay in main loop
    # main loop
    while running:
        for event in pygame.event.get():    # loop all over events
            if event.type == pygame.QUIT:
                running = False             # toggle flag to close window
        # update window
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        for object in objectList:           # draw objects in list
            object.update()                 # draw the actual object
        pygame.display.flip()               # flip double buffered window
    pygame.quit()
    quit()
```

d) Wenn Sie nun in Ihrem Hauptprogramm die Methode mainLoop() des Fensters aufrufen, werden Sie Ihr erstes Quadrat sehen! Hängen Sie weitere Quadrate in die Liste…

Aufgabe 4: Matrizendarstellung

Zur Darstellung von Conways Universe müssen wir viele Quadrate in einer Matrizendarstellung zeichnen!

a) Verwenden Sie die Methode Square, um Quadrate an verschiedenen Positionen darzustellen.
b) Schreiben Sie eine verschachtelte Schleife, mit der Sie eine Matrix darstellen können. Verwenden Sie für size die Größe eins.
c) Die richtige Skalierung der Darstellung ist nicht ganz einfach, daher benötigen wir eine Anpassung der Orthogonal-Projektion als Methode der Klasse MainWindow:

```
def OrthogonalProjection(self, xRange=(-10, 10), yRange=(-10, 10), zRange=(0, 1), frame=0):
    # set initial 2D-perspective for scene
    glMatrixMode(GL_PROJECTION) # switch to projection matrix to change settings
    glLoadIdentity()            # reset all prior projection matrices (set all to identity)
    # set scene coordinates for orthogonal projection: left, right, bottom, top, near-, far-clipping
    glOrtho(xRange[0]-frame, xRange[1]+frame, yRange[1]+frame, yRange[0]-frame, zRange[0], zRange[1])
    glMatrixMode(GL_MODELVIEW)  # switch back to model matrix to create model objects
    glLoadIdentity()            # reset all prior transformation matrices (set all to identity)
```

Diese muss mit geeigneten Parametern aufgerufen werden, bevor die Methode mainLoop() gestartet wird!
Die Parameter geben an, welche Ausmaße Ihre Zeichnung in allen 3 Dimensionen hat!