**LAPORAN PROJECT**

**PEMROGRAMAN BERORIENTASI OBJEK**

**“ PYGAME DAN KIVY ”**

****

**DISUSUN OLEH :**

**ALVIN NUR ARROFI 160411100111**

**ZAINUL ABAD 160411100116**

**DOSEN PENGAMPU :**

**ARIK KURNIAWATI, S.Kom., M.T.**

**NIP. 19780309 200312 2 009**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS TRUNOJOYO MADURA**

**2018/2019**

**DEFINISI**

**PYGAME DAN KIVY**

Pygame adalah kumpulan modul Python lintas platform yang dirancang untuk menulis video game. Ini termasuk grafik komputer dan perpustakaan suara yang dirancang untuk digunakan dengan bahasa pemrograman Python.

Kivy adalah pustaka open source Python untuk mengembangkan aplikasi seluler dan perangkat lunak aplikasi multitouch lainnya dengan antarmuka pengguna alami (NUI). Dapat berjalan di Android, iOS, Linux, OS X, dan Windows. Didistribusikan menurut syarat lisensi MIT, Kivy adalah perangkat lunak bebas dan sumber terbuka.

**INSTALLASI**

**PYGAME DAN KIVY**

* **PYGAME**

Install PyGame :

* **Pip install pygame**
* **KIVY**  
  Setelah python dipasang, buka baris perintah dan pastikan python tersedia dengan mengetik python --version. Kemudian, lakukan hal berikut untuk menginstal.

Pastikan Anda memiliki pip dan roda terbaru:

* **python -m pip install --upgrade pip wheel setuptools**

Instal dependensi (lewati gstreamer (~ 120MB) jika tidak diperlukan, lihat dependensi Kivy):

* **python -m pip menginstal docutils pygments pypiwin32 kivy.deps.sdl2 kivy.deps.glew**
* **python -m pip instal kivy.deps.gstreamer**

Catatan Jika Anda menemukan MemoryError saat menginstal, tambahkan setelah pip menginstal opsi –no-cache-dir. Untuk Python 3.5+, Anda juga dapat menggunakan backend sudut alih-alih melebur. Ini dapat diinstal dengan:

- **python -m pip instal kivy.deps.angle**

Instal kivy:

- **python -m pip instal kivy (Opsional)**

Instal contoh kivy:

- **python -m pip instal kivy\_examples**

Contoh-contoh ini dipasang di direktori share di bawah direktori root di mana python diinstal.

**PENJELASAN**

**PROGRAM**

* **PYGAME** (Ular Memakan Katak “Snacky”)

Dalam PyGame ini kami membuat game ular memakan katak yang kami beri nama game ini “Snacky” pada game ini kita bermain dengan cara ular itu memakan katak setiap makan katak ular itu bertambah poin dan bertambah Panjang, dan ketika ular itu menabrak pinggir dan menabrak badannya sendiri ular itu akan mati dan permainan mulai dari awal kembali.

* **KIVY** (Kalkulator)

Dalam Kivy ini kami membuat sebuah kalkulator, kalkulator ini seperti kalkulator seperti biasanya. Kalkulator ini bisa melakukan operasi penjumlahan , penguranga, perkalian dan pembagian.

**CODE PROGRAM**

* **PYGAME**
  + **PROGRAM START**

|  |
| --- |
| import pygame, sys  from pygame.locals import \*  WHITE = (255, 255, 255)  BLACK = (0, 0, 0)  RED = (255, 0, 0)  GREEN = (0, 255, 0)  BLUE = (0, 0, 255)  pygame.init()  pygame.display.set\_caption("!!! SNACKY !!!")  screen = pygame.display.set\_mode((960, 640), 0,32)  background = pygame.image.load('lantai1.png')  bit\_font = pygame.font.Font(('vandervon.otf'), 120)  font = pygame.font.Font(('the.ttf'), 90)  choice = pygame.Rect(320,275,320,20)  arrow = pygame.image.load('arrow.png')  clock = pygame.time.Clock()  while True:  screen.blit(background, [0, 0])  clock.tick(10)    for e in pygame.event.get():  if e.type == QUIT:  pygame.quit()  sys.exit()  elif e.type == pygame.KEYDOWN:  if e.key == pygame.K\_UP:  if choice.top == 275:  choice.move\_ip(0,150)  else:  choice.move\_ip(0,-150)  elif e.key == pygame.K\_DOWN:  if choice.top == 425:  choice.move\_ip(0,-150)  else:  choice.move\_ip(0,150)  elif e.key == pygame.K\_RETURN:  if choice.top == 275:  import snacky  elif choice.top == 425:  pygame.quit()  sys.exit()    screen.blit(bit\_font.render("'SNACKY'",False,BLUE), (230,40))  screen.blit(font.render("Play",False,BLUE), (370,250))  screen.blit(font.render("Exit",False,BLUE), (385,400))  screen.blit(arrow, choice)  pygame.display.update() |

* + **MAIN PROGRAM**

|  |
| --- |
| import pygame, sys, random  pygame.init()  X = 640  Y = 360  BLACK = (0, 0, 0)  BACKGROUND = (178, 34, 34)  WHITE = (255, 255, 255)  RED = (255, 0, 0)  YELLOW = (255, 255, 0)  BLUE = (0, 0, 255)  TRANSPARENT = (1, 2, 3)  bg = BACKGROUND  SIZE = 16  STEP = 2  fps = 100//STEP  n\_invisible = SIZE//STEP-1  n\_noCollision = SIZE//STEP \* 3  screen = pygame.display.set\_mode((960, 640), 0,32)  pygame.display.set\_caption("Pemrograman Berorientasi Object (PyGame) Abad - Alvin")  clock = pygame.time.Clock()  font = pygame.font.Font(("the.ttf"), 30)  class Snake():  def \_\_init\_\_(self, x, y, color, name):  self.startx = x  self.starty = y  self.color = color  self.name = name  def reset(self):  """ Each snake is build of several 'snakeElement's  here the list is reset to only one element """  self.snakeList = [SnakeElement(self.startx, self.starty, self.color)]  self.length = 0  self.dir = [False] \* 4 # self.dir corresponds to down, up, left, right  def grow(self):  """ One Growth Cycle consist of a number of invisible  elements, and one visible element (BLACK) """  self.length += 1  for \_ in range(n\_invisible):  self.snakeList.append(SnakeElement(self.snakeList[-1].rect.centerx,  self.snakeList[-1].rect.centery, TRANSPARENT, show=False))  self.snakeList.append(SnakeElement(self.snakeList[-1].rect.centerx,  self.snakeList[-1].rect.centery, BLACK))  def move(self):  self.move\_tail()  self.move\_head()  def move\_tail(self):  " tail moves one step forward "  for i in range(len(self.snakeList)-1, 0, -1):  self.snakeList[i].rect.center = self.snakeList[i-1].rect.center  def move\_head(self):  " Head moves according to user input (self.dir) "  head = self.snakeList[0]  down, up, left, right = self.dir  if down:  head.rect.centery += STEP  elif up:  head.rect.centery -= STEP  elif left:  head.rect.centerx -= STEP  elif right:  head.rect.centerx += STEP    def checkForCollisions(self):  head = self.snakeList[0]  ## --- check for collision with window --- ##  if (head.rect.top < 0 or  head.rect.bottom > Y or  head.rect.left < 0 or  head.rect.right > X):  game.state = game.end  return  ## --- check for collision with self --- ##  if (len(self.snakeList) > n\_noCollision and  head.rect.collidelist(self.snakeList[n\_noCollision:]) > -1):  game.state = game.end  return  ## --- check for hit with frog --- ##  if frog.rect.colliderect(head):  frog.reset()  self.grow()  def draw(self):  for s in self.snakeList:  s.draw()  txt = font.render("%s: %d" %(self.name, self.length), 1, WHITE)  screen.blit(txt, (10, 10))  class SnakeElement():  def \_\_init\_\_(self, x, y, COLOR, rad=SIZE//2, light\_pos=(0.35, -0.35), show=True):  x0 = rad + 1  y0 = rad + 1  self.surf = pygame.surface.Surface((x0\*2, y0\*2))  self.surf.fill(TRANSPARENT)  self.surf.set\_colorkey(TRANSPARENT)  self.rect = self.surf.get\_rect()  self.rect.centerx = x  self.rect.centery = y  self.show = show  if not show:  return  self.perimeter = [] # list of pixels that needs anti-aliasing  x1 = x0 + int(light\_pos[0] \* rad)  y1 = y0 + int(light\_pos[1] \* rad)  for i in range(x0\*2):  for j in range(y0\*2):  d = ((i-x0)\*\*2 + (j-y0)\*\*2)\*\*0.5  if d >= rad+1:  continue  else:  color = COLOR  # rate of color intensity change  d2 = ((i-x1)\*\*2 + (j-y1)\*\*2)\*\*0.5 \* 255 / rad  color = gradient(d2, color)  if d > rad: # prepare for anti-aliasing, get the perimeter pixels  alfa = d-rad  color\_rim = [c \* (1-alfa) for c in color]  self.perimeter.append(((i, j), color\_rim, alfa))  color = TRANSPARENT # colorkey at rim  self.surf.set\_at((i, j), color)  def draw(self):  if self.show is False:  return    screen.blit(self.surf, self.rect)  # anti-aliasing  for p in self.perimeter:  x, y = p[0]  x += self.rect.x  y += self.rect.y  if x < 0 or x >= X or y < 0 or y >= Y:  continue  color = p[1]  alfa = p[2]  bg = screen.get\_at((x, y))  color\_aa = [rim + back \* alfa for rim, back in zip(color, bg)]  screen.set\_at((x, y), color\_aa)  class Frog():  def \_\_init\_\_(self):  self.surf = pygame.surface.Surface((SIZE, SIZE))  self.surf.fill(bg)  self.surf.blit(picFrog, [0, 0])  self.rect = self.surf.get\_rect()  def draw(self):  screen.blit(self.surf, self.rect)  def reset(self):  self.rect.centerx = random.randint(SIZE//2, X-SIZE//2)  self.rect.centery = random.randint(30, Y-SIZE//2)  while self.rect.collidelist(snake.snakeList) > -1:  self.rect.centerx = random.randint(SIZE//2, X-SIZE//2)  self.rect.centery = random.randint(30, Y-SIZE//2)  class GameState():  def \_\_init\_\_(self):  self.state = self.gameLoop  def gameLoop(self):  clock.tick(fps) # game speed  screen.fill(bg)  self.eventLoop()  snake.move()  snake.checkForCollisions()  frog.draw()  snake.draw()  pygame.display.flip()  def end(self):  againTxt()  self.state = self.playAgain  def stop(self):  pygame.quit()  sys.exit()  def playAgain(self):  pygame.time.wait(100)  for event in pygame.event.get():  if event.type == pygame.QUIT:  self.state = self.stop  return  elif event.type == pygame.KEYDOWN:  if event.key == pygame.K\_n:  self.state = self.stop  return  elif event.key == pygame.K\_y:  snake.reset()  frog.reset()  self.state = self.gameLoop  return  def eventLoop(self):  for event in pygame.event.get():  if event.type == pygame.QUIT:  self.state = self.stop  return  elif event.type == pygame.KEYDOWN:  if event.key == pygame.K\_DOWN and not snake.dir[1]:  snake.dir = [True, False, False, False]  elif event.key == pygame.K\_UP and not snake.dir[0]:  snake.dir = [False, True, False, False]  elif event.key == pygame.K\_LEFT and not snake.dir[3]:  snake.dir = [False, False, True, False]  elif event.key == pygame.K\_RIGHT and not snake.dir[2]:  snake.dir = [False, False, False, True]  def gradient(d2, color):  r, g, b = color  return [max(255-int(d2\*(255-r)//150), 0),  max(255-int(d2\*(255-g)//150), 0),  max(255-int(d2\*(255-b )//150), 0)]  def againTxt():  txt = font.render("Snake Dies", 1, RED)  rct = txt.get\_rect(center=[X//1-150, Y//1-50])  screen.blit(txt, rct)  txt = font.render("Main Lagi? (Y / N)?", 1, WHITE)  rct = txt.get\_rect(center=[X//1-150, Y//1])  screen.blit(txt, rct)  pygame.display.flip()  picFrog = pygame.image.load("frog.png").convert()  picFrog = pygame.transform.scale(picFrog, (SIZE, SIZE))  picFrog.set\_colorkey(WHITE)  snake = Snake(X//2, Y//3, YELLOW, 'Yellow Player ')  frog = Frog()  game = GameState()  snake.reset()  frog.reset()  while True:  game.state() |

* **KIVY**
  + **MAIN PROGRAM**

|  |
| --- |
| import kivy  kivy.require("1.9.0")    from kivy.app import App  from kivy.uix.gridlayout import GridLayout    class CalcGridLayout(GridLayout):    # Fungsi berikut akan dipanggil saat tombol "=" di tekan  def calculate(self, calculation):  if calculation:  try:  # Menyelesaikan hitungan yang ada di entry  # yang mana nantinya akan di tampilkan  self.display.text = str(eval(calculation))  except Exception:  self.display.text = "Error"    class CalculatorApp(App):    def build(self):  return CalcGridLayout()    calcApp = CalculatorApp()  calcApp.run() |

* + **CALCULATOR.KV**

|  |
| --- |
| <CustButton@Button>:  font\_size: 32  <CalcGridLayout>:  id: calculator  display: entry  rows: 5  padding: 10  spacing: 10  # Tampilan saat menginput  BoxLayout:  TextInput:  id: entry  font\_size: 32  multiline: False  # Saat tombol di tekan, input akan ditampilkan di entry  BoxLayout:  spacing: 10  CustButton:  text: "AC"  on\_press: entry.text = ""  CustButton:  text: "7"  on\_press: entry.text += self.text  CustButton:  text: "8"  on\_press: entry.text += self.text  CustButton:  text: "9"  on\_press: entry.text += self.text  BoxLayout:  spacing: 10  CustButton:  text: "+"  on\_press: entry.text += self.text  CustButton:  text: "4"  on\_press: entry.text += self.text  CustButton:  text: "5"  on\_press: entry.text += self.text  CustButton:  text: "6"  on\_press: entry.text += self.text  BoxLayout:  spacing: 10  CustButton:  text: "-"  on\_press: entry.text += self.text  CustButton:  text: "1"  on\_press: entry.text += self.text  CustButton:  text: "2"  on\_press: entry.text += self.text  CustButton:  text: "3"  on\_press: entry.text += self.text  # Saat "=" di tekan di entry  # maka akan menuju ke fungsi calculate  BoxLayout:  spacing: 10  CustButton:  text: "\*"  on\_press: entry.text += self.text  CustButton:  text: "/"  on\_press: entry.text += self.text  CustButton:  text: "0"  on\_press: entry.text += self.text  CustButton:  text: "="  on\_press: calculator.calculate(entry.text) |

**PENJELASAN**

**CODE PROGRAM**

**“OBJECT-ORIENTED PROGRAMMING (OOP)”**

* **PYGAME**

Class Frog

* Draw
* reset

Class Snake

* reset
* grow
* move
* move\_tail
* move\_head
* checkforcollision
* draw

Class GameState

* gameLoop
* end
* stop
* playAgain
* eventLoop
* gradient
* againTxt

Class SnakeElement

* draw

**kelas Snake**   
Kelas Snake terdiri dari sejumlah metode.

* *def \_\_* *init* *(* *):*   
  Dalam def \_\_ init \_\_ () posisi awal ditetapkan. Warnanya adalah warna kepala. Nama ini terutama persiapan untuk versi dua pemain berikutnya.
* *def reset ():*   
  Dalam metode reset (), SnakeElement (kepala) pertama didefinisikan dan ditempatkan dalam daftar, ' snakeList '.   
  Panjang (tail-) adalah nol dan semua gerakan dihentikan - daftar self.dir memiliki 4 elemen Boolean yang menunjukkan gerakan ke arah bawah, atas, kiri atau kanan; mereka semua di atur ke 'Salah'.
* *def tumbuh ():*   
  Dalam metode grow (), sebuah angka (yang didefinisikan oleh ' n\_invisible ') dari elemen-elemen tubuh yang tidak terlihat ditambahkan ke snakeList dan akhirnya satu elemen tubuh yang terlihat (HITAM) ditambahkan. Semua elemen baru terletak di posisi elemen yang ada terakhir - selama beberapa gerakan berikutnya langkah-langkah elemen ekor baru secara bertahap akan terlihat.   
  Jumlah elemen tak terlihat bergantung pada SIZE dan LANGKAH, lihat di bawah ini di bawah kode utama.
* *def move ():*   
  Dalam langkah () kita cukup memanggil metode move\_tail () dan move\_head ().
* *def* *move\_tail* *():*   
  Dalam for-loop yang dimulai pada ujung ekor, elemen-elemen ekor dipindahkan ke posisi elemen sebelumnya.
* def move\_head ():   
  Garis "kepala = self.snakeList [0]" mengurangi pengetikan berikutnya dan meningkatkan keterbacaan.   
  Self.dir diperluas ke variabel Boolean 'down', 'up', 'kiri', dan 'benar'; salah satunya akan Benar, yang lain akan Salah.   
  Akhirnya, posisi kepala dipindahkan sesuai dengan arah self.dir .
* *def* *checkForCollision* *():*   
  Metode ini menangani tiga situasi tabrakan yang berbeda.   
    
  Pemeriksaan pertama untuk tabrakan dengan batas layar, ini tidak perlu penjelasan lebih lanjut.   
    
  Pemeriksaan kedua adalah tabrakan dengan ekor. Ketika elemen-elemen tubuh melingkar ditempatkan di permukaan persegi, tidak dapat dihindarkan bahwa elemen permukaan saling tumpang tindih ketika kepala berputar. Dengan demikian, kita perlu mengabaikan tabrakan antara kepala dan beberapa elemen tubuh pertama.   
  Hanya jika jumlah elemen-tubuh melebihi ambang batas, kami memeriksa tabrakan dan hanya untuk elemen dengan jumlah yang lebih tinggi dari ambang ini. Ambang batas didefinisikan dalam main ( parameter ' n\_noCollision '); Saya telah memilih 'zona bebas' dari tiga elemen tubuh (dua elemen ekor).   
    
  Pemeriksaan ketiga adalah tabrakan dengan 'katak' di mana katak.menet () dan self.grow () dipanggil.
* *draw ():*   
  Kami mengulang melalui snakeList dan memanggil fungsi draw () dari masing-masing body-element, keduanya terlihat dan tidak terlihat.   
  Sebuah teks kecil dirender dan di- bled ke layar.

**kelas** **SnakeElement**   
Kelas SnakeElement hanya memiliki dua metode, metode \_\_ init \_\_ () yang besar dan metode draw () yang lebih kecil.   
  
Pada dasarnya, SnakeElement mendefinisikan dan menarik salah satu dari dua jenis elemen grafis, elemen-tubuh yang terlihat atau yang tidak terlihat.   
Dalam panggilan ke SnakeElement , parameter 'show' dapat disetel ke True (= visible) atau False (= tak terlihat).   
Unsur-unsur tak terlihat hanya terdiri dari spesifikasi Permukaan dasar dan Rektum .   
Pada metode draw (), kita pertama menguji

jika pertunjukkan adalah False Kembali

**kelas Apple ()**   
Kelas ini memiliki metode \_\_ init \_\_ () dan draw () yang sederhana; mereka tidak memerlukan penjelasan lebih lanjut.   
  
Metode reset () digunakan baik pada inisialisasi pertama dan setelah dipukul dengan Snake; itu mendefinisikan posisi acak dan memastikan bahwa posisi ini tidak bertabrakan dengan ular yang ada - jika posisi baru ditemukan.   
  
**kelas** **GameState** **()**   
Untuk penjelasan mendalam dari kelas GameSate,   
  
The GameState didasari dari sejumlah metode kecil.

* *def* *eventLoop* *():*   
  The eventLoop () menangani input pengguna dari keyboard.   
    
  Karena Snake tidak dapat mengubah 180 derajat (ia akan langsung bertabrakan dengan dirinya), semua input arah diperiksa untuk arah sebaliknya menjadi aktif; hanya jika ini tidak terjadi, arah baru diimplementasikan dalam daftar snake.dir .   
    
  **Fungsi umum**   
  Dua fungsi umum kecil digunakan: gradient () dan againTxt ().   
  Metode gradien () dijelaskan dalam tutorial antialiasing yang disebutkan di atas.   
  The againTxt () menempatkan teks sederhana di layar jika ular mati.

**Kode utama**   
  
Saya telah menempatkan semua konstanta dan deklarasi Pygame dasar di bagian atas kode. Semoga sebagian besar dari hal ini dikenal.    
Nilai ' n\_invisible ' dan ' n\_noCollision ' dihitung menggunakan dua rumus sederhana.   
  
Di bagian bawah kode saya pertama kali mengimpor dan mengubah gambar katak (file terlampir).   
  
Kemudian tiga instance kelas dinyatakan: ular, katak, dan game.   
  
snake.reset () dan apple.reset () menginisialisasi posisi ular dan katak.   
  
Akhirnya, game-loop utama yang sederhana dijelaskan dalam tutorial permainan-loop. 

* **KIVY**

Class CalculatorApp(App)

* build

Class CalcGridLayout(GridLayout)

* calculate

Kelas CalcGridLayout : kelas ini berfungsi untuk mengatur widget di grid. Anda harus menentukan setidaknya satu dimensi dari grid sehingga kivy dapat menghitung ukuran elemen dan bagaimana untuk mengatur mereka.

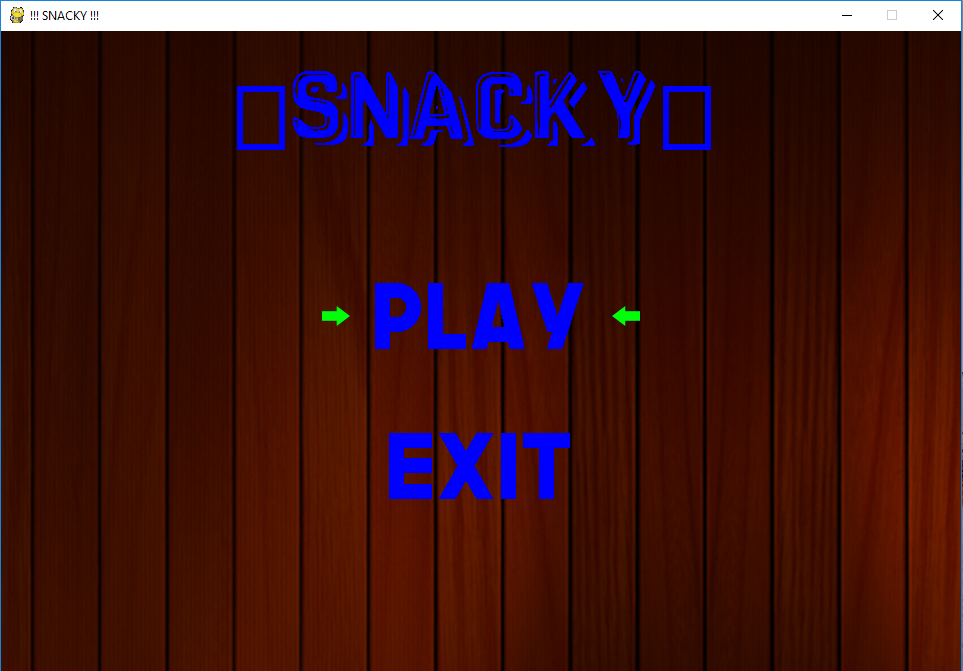
def calculate : fungsi yang akan menghitung operasi yang diberikan

Kelas CalculatorApp : kelas ini berfungsi untuk menjalankan aplikasi. Di dalam kelas ini hanya terdapat 1 method, yaitu method build untuk menjalankan aplikasi CalculatorApp denganmemanggil class CalcGridLayout**.**

**HASIL**

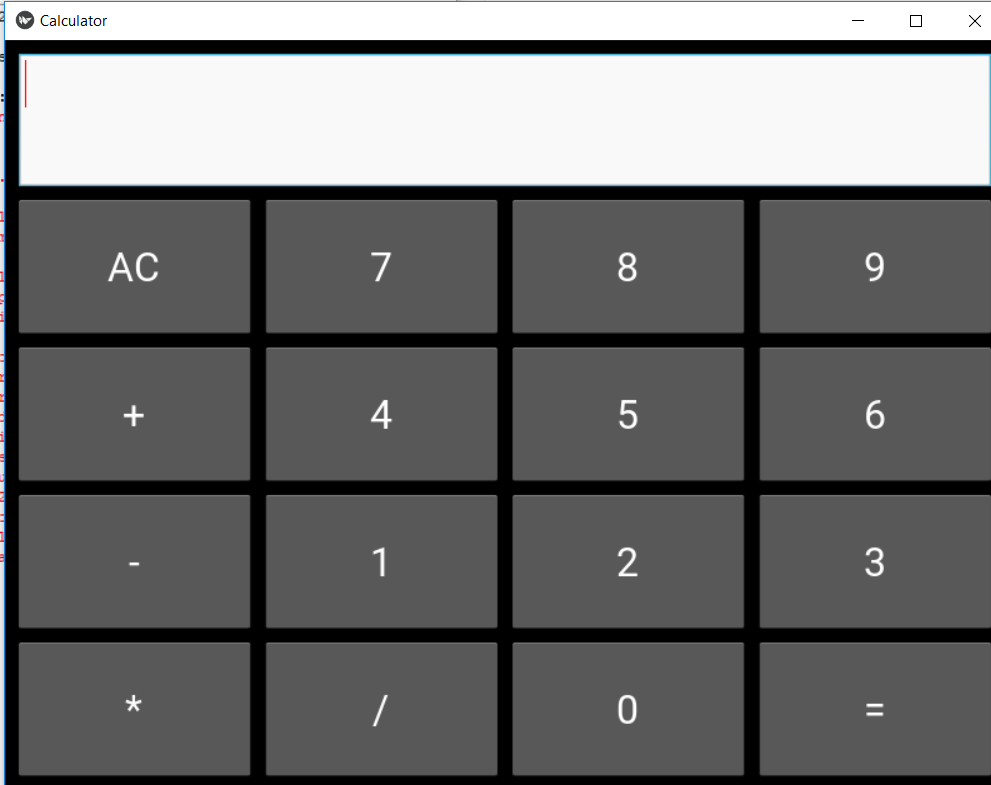
**RUNNING PROGRAM**

* **PYGAME**
  + **START PROGRAM**



* + **MAIN PROGRAM**



* **KIVY**
  + **MAIN PROGRAM**

**SOURCE**

**PROGRAM**

* **PYGAME**

<https://www.dreamincode.net/forums/topic/402466-snake-game-a-pygame-tutorial/>

|  |
| --- |
| """  A classic Single Player Snake Game  By DK3250, March 2017  This game is made as a tutorial for  DreamInCode's Python Forum  It demonstrates a vararity of Pygame functions  as well as basic Python programming.  The comments are few as the accompanoying text  aer meant to explain the code.  The code is made with Python 3.4 and Pygame 1.9  """  import pygame, sys, random  pygame.init()  X = 400  Y = 300  BLACK = (0, 0, 0)  GREEN = (0, 255, 0)  WHITE = (255, 255, 255)  RED = (255, 0, 0)  YELLOW = (255, 255, 0)  BLUE = (0, 0, 255)  TRANSPARENT = (1, 2, 3)  bg = GREEN  SIZE = 16  STEP = 2  fps = 100//STEP  n\_invisible = SIZE//STEP-1  n\_noCollision = SIZE//STEP \* 3  screen = pygame.display.set\_mode((X, Y))  pygame.display.set\_caption("Snake Game Presentation by DK3250")  clock = pygame.time.Clock()  font = pygame.font.SysFont("Verdana", 18)  class Snake():  """ The basic snake object  Snake is build of snakeElement's - they are separate objects.  The methods in Snake are:  \* \_\_init\_\_()  \* reset()  \* grow()  \* move()  \* move\_tail()  \* move\_head()  \* checkForCollisions()  \* draw()  See the tutorial text for explanation.  """  def \_\_init\_\_(self, x, y, color, name):  self.startx = x  self.starty = y  self.color = color  self.name = name  def reset(self):  """ Each snake is build of several 'snakeElement's  here the list is reset to only one element """  self.snakeList = [SnakeElement(self.startx, self.starty, self.color)]  self.length = 0  self.dir = [False] \* 4 # self.dir corresponds to down, up, left, right  def grow(self):  """ One Growth Cycle consist of a number of invisible  elements, and one visible element (BLACK) """  self.length += 1  for \_ in range(n\_invisible):  self.snakeList.append(SnakeElement(self.snakeList[-1].rect.centerx,  self.snakeList[-1].rect.centery, TRANSPARENT, show=False))  self.snakeList.append(SnakeElement(self.snakeList[-1].rect.centerx,  self.snakeList[-1].rect.centery, BLACK))  def move(self):  self.move\_tail()  self.move\_head()  def move\_tail(self):  " tail moves one step forward "  for i in range(len(self.snakeList)-1, 0, -1):  self.snakeList[i].rect.center = self.snakeList[i-1].rect.center  def move\_head(self):  " Head moves according to user input (self.dir) "  head = self.snakeList[0]  down, up, left, right = self.dir  if down:  head.rect.centery += STEP  elif up:  head.rect.centery -= STEP  elif left:  head.rect.centerx -= STEP  elif right:  head.rect.centerx += STEP    def checkForCollisions(self):  head = self.snakeList[0]  ## --- check for collision with window --- ##  if (head.rect.top < 0 or  head.rect.bottom > Y or  head.rect.left < 0 or  head.rect.right > X):  game.state = game.end  return  ## --- check for collision with self --- ##  if (len(self.snakeList) > n\_noCollision and  head.rect.collidelist(self.snakeList[n\_noCollision:]) > -1):  game.state = game.end  return  ## --- check for hit with apple --- ##  if apple.rect.colliderect(head):  apple.reset()  self.grow()  def draw(self):  for s in self.snakeList:  s.draw()  txt = font.render("%s: %d" %(self.name, self.length), 1, WHITE)  screen.blit(txt, (10, 10))  class SnakeElement():  """  Creation of an anti-aliased snake body-element.  In essence a circle with a color gradient and anti-aliased to the background.    Both the gardient function (see below) and the anti-aliasing is explained here:  http://www.dreamincode.net/forums/topic/388729-anti-aliasing/  """  def \_\_init\_\_(self, x, y, COLOR, rad=SIZE//2, light\_pos=(0.35, -0.35), show=True):  x0 = rad + 1  y0 = rad + 1  self.surf = pygame.surface.Surface((x0\*2, y0\*2))  self.surf.fill(TRANSPARENT)  self.surf.set\_colorkey(TRANSPARENT)  self.rect = self.surf.get\_rect()  self.rect.centerx = x  self.rect.centery = y  self.show = show  if not show:  return  self.perimeter = [] # list of pixels that needs anti-aliasing  x1 = x0 + int(light\_pos[0] \* rad)  y1 = y0 + int(light\_pos[1] \* rad)  for i in range(x0\*2):  for j in range(y0\*2):  d = ((i-x0)\*\*2 + (j-y0)\*\*2)\*\*0.5  if d >= rad+1:  continue  else:  color = COLOR  # rate of color intensity change  d2 = ((i-x1)\*\*2 + (j-y1)\*\*2)\*\*0.5 \* 255 / rad  color = gradient(d2, color)  if d > rad: # prepare for anti-aliasing, get the perimeter pixels  alfa = d-rad  color\_rim = [c \* (1-alfa) for c in color]  self.perimeter.append(((i, j), color\_rim, alfa))  color = TRANSPARENT # colorkey at rim  self.surf.set\_at((i, j), color)  def draw(self):  """  First, the self.surf (picture without perimeter) is blitted to screen.  Secondly, the perimeter is updated, anti-aliasing to the background.  """  if self.show is False:  return    screen.blit(self.surf, self.rect)  # anti-aliasing  for p in self.perimeter:  x, y = p[0]  x += self.rect.x  y += self.rect.y  if x < 0 or x >= X or y < 0 or y >= Y:  continue  color = p[1]  alfa = p[2]  bg = screen.get\_at((x, y))  color\_aa = [rim + back \* alfa for rim, back in zip(color, bg)]  screen.set\_at((x, y), color\_aa)  class Apple():  def \_\_init\_\_(self):  self.surf = pygame.surface.Surface((SIZE, SIZE))  self.surf.fill(bg)  self.surf.blit(picApple, [0, 0])  self.rect = self.surf.get\_rect()  def draw(self):  screen.blit(self.surf, self.rect)  def reset(self):  self.rect.centerx = random.randint(SIZE//2, X-SIZE//2)  self.rect.centery = random.randint(30, Y-SIZE//2)  while self.rect.collidelist(snake.snakeList) > -1:  self.rect.centerx = random.randint(SIZE//2, X-SIZE//2)  self.rect.centery = random.randint(30, Y-SIZE//2)  class GameState():  def \_\_init\_\_(self):  self.state = self.gameLoop  def gameLoop(self):  """ The basic game loop handling user input during game """  clock.tick(fps) # game speed  screen.fill(bg)  self.eventLoop()  snake.move()  snake.checkForCollisions()  apple.draw()  snake.draw()  pygame.display.flip()  def end(self):  """ This method is called when a game ends i.e. one snake dies """  againTxt()  self.state = self.playAgain  def stop(self):  """ This method completely stops the game """  pygame.quit()  sys.exit()  def playAgain(self):  """ Loop to handle keyboard input for 'State = Play Again'  defined in the method 'End' """  pygame.time.wait(100)  for event in pygame.event.get():  if event.type == pygame.QUIT:  self.state = self.stop  return  elif event.type == pygame.KEYDOWN:  if event.key == pygame.K\_n:  self.state = self.stop  return  elif event.key == pygame.K\_y:  snake.reset()  apple.reset()  self.state = self.gameLoop  return  def eventLoop(self):  """ Event Loop, - handles user input """  for event in pygame.event.get():  if event.type == pygame.QUIT:  self.state = self.stop  return  elif event.type == pygame.KEYDOWN:  if event.key == pygame.K\_DOWN and not snake.dir[1]:  snake.dir = [True, False, False, False]  elif event.key == pygame.K\_UP and not snake.dir[0]:  snake.dir = [False, True, False, False]  elif event.key == pygame.K\_LEFT and not snake.dir[3]:  snake.dir = [False, False, True, False]  elif event.key == pygame.K\_RIGHT and not snake.dir[2]:  snake.dir = [False, False, False, True]  def gradient(d2, color):  """  A general color gradient function.  With a color input and distance input  a gradient color is calculated.  """  r, g, b = color  return [max(255-int(d2\*(255-r)//150), 0),  max(255-int(d2\*(255-g)//150), 0),  max(255-int(d2\*(255-b )//150), 0)]  def againTxt():  """ Text at game end (snake dead) """  txt = font.render("Snake Dies", 1, RED)  rct = txt.get\_rect(center=[X//2, Y//2])  screen.blit(txt, rct)  txt = font.render("Play again (y/n)?", 1, WHITE)  rct = txt.get\_rect(center=[X//2, Y//2+30])  screen.blit(txt, rct)  pygame.display.flip()  picApple = pygame.image.load("picture.png").convert()  picApple = pygame.transform.scale(picApple, (SIZE, SIZE))  picApple.set\_colorkey(WHITE)  snake = Snake(X//2, Y//3, YELLOW, 'Yellow Player ')  apple = Apple()  game = GameState()  snake.reset()  apple.reset()  while True:  game.state() |

* **KIVY**

<http://www.newthinktank.com/2016/10/kivy-tutorial-3/>

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | # ---------- main.py  ----------    import kivy  kivy.require("1.9.0")    from kivy.app import App  from kivy.uix.gridlayout import GridLayout    class CalcGridLayout(GridLayout):        # Function called when equals is pressed      def calculate(self, calculation):          if calculation:              try:                  # Solve formula and display it in entry                  # which is pointed at by display                  self.display.text = str(eval(calculation))              except Exception:                  self.display.text = "Error"    class CalculatorApp(App):        def build(self):          return CalcGridLayout()    calcApp = CalculatorApp()  calcApp.run()    # ---------- calculator.kv  ----------    # Custom button  <CustButton@Button>:      font\_size: 32    # Define id so I can refer to the CalcGridLayout  # class functions  # Display points to the entry widget  <CalcGridLayout>:      id: calculator      display: entry      rows: 5      padding: 10      spacing: 10        # Where input is displayed      BoxLayout:          TextInput:              id: entry              font\_size: 32              multiline: False        # When buttons are pressed update the entry      BoxLayout:          spacing: 10          CustButton:              text: "7"              on\_press: entry.text += self.text          CustButton:              text: "8"              on\_press: entry.text += self.text          CustButton:              text: "9"              on\_press: entry.text += self.text          CustButton:              text: "+"              on\_press: entry.text += self.text        BoxLayout:          spacing: 10          CustButton:              text: "4"              on\_press: entry.text += self.text          CustButton:              text: "5"              on\_press: entry.text += self.text          CustButton:              text: "6"              on\_press: entry.text += self.text          CustButton:              text: "-"              on\_press: entry.text += self.text        BoxLayout:          spacing: 10          CustButton:              text: "1"              on\_press: entry.text += self.text          CustButton:              text: "2"              on\_press: entry.text += self.text          CustButton:              text: "3"              on\_press: entry.text += self.text          CustButton:              text: "\*"              on\_press: entry.text += self.text        # When equals is pressed pass text in the entry      # to the calculate function      BoxLayout:          spacing: 10          CustButton:              text: "AC"              on\_press: entry.text = ""          CustButton:              text: "0"              on\_press: entry.text += self.text          CustButton:              text: "="              on\_press: calculator.calculate(entry.text)          CustButton:              text: "/"              on\_press: entry.text += self.text | |