|  |  |  |
| --- | --- | --- |
|  | Національний університет «Львівська політехніка»  **2023/2024** навчальний рік  **Весняний** семестр | **Розробка кросплатформених додатків (Python)**  **Розрахунково-графічна робота:**  Візуалізація явищ та процесів засобами стандартної бібліотеки turtle |
| **Кафедра** систем автоматизованого проектування  **Викладач:** Гавран В.Б. | **Група:** ПП-24  **Студент:** Верещак Богдан |
| Мета роботи | | |
| Розробити візуалізацію вказаного явища або процесу засобами стандартної бібліотеки turtle мови. | | |
| Завдання | | |
| 1. Побудувати засобами turtle фоновий статичний рисунок, який відповідає обставинам вибраного явища або процесу **[1 бал]**.  2. Анімувати засобами turtle вибране явище або процес із врахуванням його фізики **[1 бал]**.  3. Забезпечити засобами turtle інтерактивну взаємодію із налаштування одного параметра або початкової умови для вибраного явища або процесу **[1 бал].**  4. Виконати візуалізацію вказаного явища або процесу для кількох значень досліджуваного параметра або початкової умови **[1 бал]**. Моя тема для розрахунково-графічної роботи: Інтерактивний симулятор фізики м’яча | | |
| 1. Програма мовою Python | | |
| **Файл ball.py:**  import turtle  import math  from ground import \*  from line import \*  class Ball:  def \_\_init\_\_(self, x, y, vx, vy, radius):  self.x = x  self.y = y  self.vx = vx  self.vy = vy  self.radius = radius  self.active = True  def draw(self):  turtle.penup()  turtle.goto(self.x, self.y - self.radius)  turtle.pendown()  turtle.circle(self.radius)  def move(self):  self.vy -= 0.5 # Гравітація  self.x += self.vx  self.y += self.vy  def check\_collision(self, surface):  ground\_y = surface.get\_y(self.x)  if self.y - self.radius <= ground\_y:  self.y = ground\_y + self.radius  normal\_angle = surface.get\_normal\_angle(self.x)  self.bounce(normal\_angle)  # Відбиття від лівої та правої стін екрану  if self.x + self.radius >= 600 or self.x - self.radius <= -600:  self.vx = -self.vx \* 0.95  self.x = max(min(self.x, 600 - self.radius), -600 + self.radius)    for line in lines:  if self.line\_collision(line):  normal\_angle = self.get\_line\_normal\_angle(line)  self.bounce(normal\_angle)  break  def line\_collision(self, line):  # Compute the closest point on the line segment to the ball center  line\_vec = (line.x2 - line.x1, line.y2 - line.y1)  pnt\_vec = (self.x - line.x1, self.y - line.y1)  line\_len = math.sqrt(line\_vec[0]\*\*2 + line\_vec[1]\*\*2)  line\_unitvec = (line\_vec[0] / line\_len, line\_vec[1] / line\_len)  proj\_len = pnt\_vec[0] \* line\_unitvec[0] + pnt\_vec[1] \* line\_unitvec[1]  closest\_pnt = (line.x1 + proj\_len \* line\_unitvec[0], line.y1 + proj\_len \* line\_unitvec[1])  # Clamp the closest point to the line segment  closest\_pnt = (max(min(closest\_pnt[0], max(line.x1, line.x2)), min(line.x1, line.x2)),  max(min(closest\_pnt[1], max(line.y1, line.y2)), min(line.y1, line.y2)))  dist\_to\_line = math.sqrt((closest\_pnt[0] - self.x)\*\*2 + (closest\_pnt[1] - self.y)\*\*2)  return dist\_to\_line <= self.radius    def get\_line\_normal\_angle(self, line):  line\_angle = math.atan2(line.y2 - line.y1, line.x2 - line.x1)  normal\_angle = line\_angle + math.pi / 2  return normal\_angle  def bounce(self, normal\_angle):  speed = math.sqrt(self.vx\*\*2 + self.vy\*\*2)  incidence\_angle = math.atan2(self.vy, self.vx)  reflection\_angle = normal\_angle + incidence\_angle  self.vx = speed \* math.cos(reflection\_angle) \* 0.9 # Зменшення швидкості для врахування втрат енергії  self.vy = speed \* math.sin(reflection\_angle) \* 0.9  if abs(self.vx) < 0.1: # Зупиняємо м'яч, якщо швидкість дуже мала  self.vy = 0  self.vx = 0  self.active = False  **Файл ground.py:**  import turtle  import random  import math  t = turtle  class Surface:  def \_\_init\_\_(self):  self.points = []  self.generate\_terrain()  def generate\_terrain(self):  x = -600  y=-300  while x <= 600:  y = random.randint(max(y-20,-300),y+20)  self.points.append((x, y))  x += 20  def draw(self):  t.hideturtle()  t.fillcolor('#8C6F3E')  t.begin\_fill()  t.teleport(-600, -400)  for point in self.points:  turtle.goto(point)  turtle.pendown()  t.setposition(point)  t.setposition(600,-400)  t.end\_fill()  def get\_y(self, x):  for i in range(len(self.points) - 1):  if self.points[i][0] <= x <= self.points[i+1][0]:  x1, y1 = self.points[i]  x2, y2 = self.points[i+1]  return y1 + (y2 - y1) \* (x - x1) / (x2 - x1)  return 0  def get\_normal\_angle(self, x):  for i in range(len(self.points) - 1):  if self.points[i][0] <= x <= self.points[i+1][0]:  x1, y1 = self.points[i]  x2, y2 = self.points[i+1]  angle = math.atan2(y2 - y1, x2 - x1)  return angle + math.pi / 2  return math.pi / 2  **Файл line.py:**  import turtle  import math  lines = []  class Line:  def \_\_init\_\_(self,x1,y1,x2,y2) -> None:  self.x1=x1  self.y1=y1  self.x2=x2  self.y2=y2  def draw(self):  turtle.penup()  turtle.goto(self.x1, self.y1)  turtle.dot(3)  turtle.pendown()  turtle.goto(self.x2, self.y2)  turtle.dot(3)  turtle.penup()  **Файл main.py:**  import turtle  import random  import time  from ball import \*  from ground import \*  from line import \*  # Основний код програми  screen = turtle.Screen()  screen.setup(width=1200, height=800)  screen.tracer(0)  screen.title('Гравітація з шаріками')  screen.bgcolor('#9dd3f2')  turtle.color("#914a03")  radius = 10  sleep = 0.001  first\_click = True  xpos1=None  ypos1=None  xpos2=None  ypos2=None  surface = Surface()  balls = []  def create\_ball(x, y, radius=10):  vx = random.uniform(-5, 5)  vy = random.uniform(5, 15)  ball = Ball(x, y, vx, vy, radius)  balls.append(ball)  def on\_right\_click(x, y):  create\_ball(x, y, radius)  def create\_line(x1,y1,x2,y2):  line = Line(x1,y1,x2,y2)  lines.append(line)  line.draw()  def on\_left\_click(x,y):  global first\_click, xpos1, ypos1,xpos2,ypos2  if first\_click:  xpos1 = x  ypos1 = y  first\_click = False  else:  xpos2 = x  ypos2 = y  create\_line(xpos1,ypos1,xpos2,ypos2)  first\_click = True  print(xpos1,ypos1,xpos2,ypos2)  def on\_a\_click():  global radius  radius += 1  def on\_d\_click():  global radius  radius -= 1  def on\_z\_click():  global sleep  sleep += 0.001  def on\_c\_click():  global sleep  if sleep>=0:  return  sleep -= 0.001  screen.onscreenclick(on\_right\_click, 3)  screen.onscreenclick(on\_left\_click, 1)  screen.onkeypress(on\_a\_click,"a")  screen.onkeypress(on\_d\_click,"d")  screen.onkeypress(on\_z\_click,"z")  screen.onkeypress(on\_c\_click,"c")  screen.listen()  while True:  turtle.clear()  surface.draw()  for ball in balls:  if ball.active:  ball.move()  ball.check\_collision(surface)  ball.draw()  for line in lines:  line.draw()  screen.update()  time.sleep(sleep)  turtle.done() | | |
| 2. Знімок екрану із візуалізацією вказаного явища або процесу | | |
|  | | |
| **3. Опис отриманих результатів** | | |
| Ця програма написана на Python із використанням бібліотеки turtle для створення симуляції м'ячів, які відбиваються від нерівної поверхні, з можливістю створення м’ячів та платформ.  Поверхня (Surface):   * Генерує нерівну поверхню шляхом випадкового вибору координат y для кожної точки x. * Малює поверхню, з'єднуючи всі згенеровані точки. * Обчислює y-координату для заданої x-координати (для визначення рівня землі в будь-якій точці). * Обчислює нормальний кут поверхні в заданій точці (для відбиття м'ячів).   М'яч (Ball):   * Ініціалізується з позицією (x, y), швидкістю (vx, vy), радіусом та масою. * Малює м'яч на екрані. * Рухається з врахуванням гравітації. * Перевіряє зіткнення з поверхнею, відбивається від поверхні та країв екрану. * Використовує фізичні закони для відбиття від поверхні та обчислення.   Основний код:   * Створює поверхню та ініціалізує список м'ячів. * Обробляє події натискання миші для створення нових м'ячів. * Обробляє події натискання клавіш для зміни радіуса та маси м'ячів. * Оновлює екран в циклі, малює поверхню та м'ячі, оновлює їхні позиції та обробляє зіткнення.   Лінія (Line):   * Ініціалізує позицію двох точок * Малює лінію на екрані   Керування програмою здійснюється за допомогою клавіш клавіатури. Кожна клавіша відповідає певній дії з об'єктом:   * RightClick – Спавн м’яча * LeftClick – Поставити першу і другу позицію для генерації прямої * a – збільшення радіусів для майбутнії мячів * d – зменшення радіусів для майбутнії мячів * z – сповільнює програму * с – уповільнює програму   Програма демонструє, як зміна параметрів, таких як радіус, та створення додаткових перешкод може впливати на результати зіткнень та відбиттів, і надає користувачам можливість взаємодіяти з симуляцією в реальному часі. | | |