1. 공을 ball2.png로 변경

- tkinter.PhotoImage, Canvas.create\_image 사용

- 그림의 Canvas.coords의 결과는 중심 좌표

. 이전 소스에서 사용된 oval의 coords는 left, top, right, bottom

- 방법1)

. 좌표 사용되는 위치에서 모두 수정

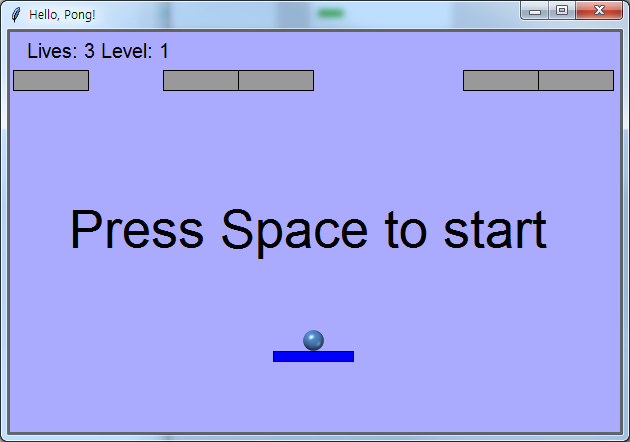
. Ball.\_\_init\_\_, Ball.update, Ball.collide, Game.check\_collisions 수정

- 방법2)

. Ball.get\_position을 overriding

|  |
| --- |
| Description]  get\_position 함수를 overriding함 |
| Code]  def \_\_init\_\_(self, canvas, x, y):  self.radius = 10  self.direction = [1, -1]  self.speed = 10  self.ballimage = tk.PhotoImage(file='ball2.png') # PhotoImage를 멤버변수로 저장하지 않으면 사진이 할당해제되어 이미지가 보이지 않음  image = canvas.create\_image(x, y, anchor=tk.CENTER, image=self.ballimage)  super(Ball, self).\_\_init\_\_(canvas, image  def get\_position(self):  center = self.canvas.coords(self.item)  left = center[0] - self.radius  right = center[0] + self.radius  top = center[1] - self.radius  bottom = center[1] + self.radius  return [left, top, right, bottom] |
| Game Shot] |

2. Level 추가, 벽돌 층수 변경, 임의의 벽돌만 생성



- Game.\_\_int\_\_에서 self.level = 1로 초기화

- Game.game\_loop에서 num\_bricks == 0인 경우에 level 증가, <space> binding 등의 추가 필요

- Game.update\_lives\_text에 level 표시: ‘%d %d’ % (x, y)

- Level 시작할 때, 벽돌 생성을 함수로 변경

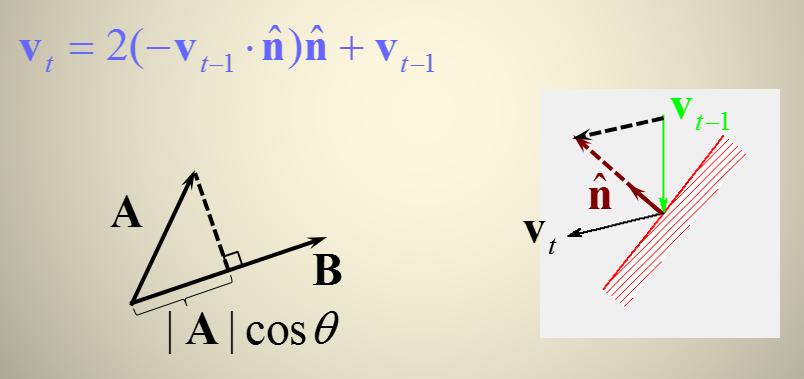
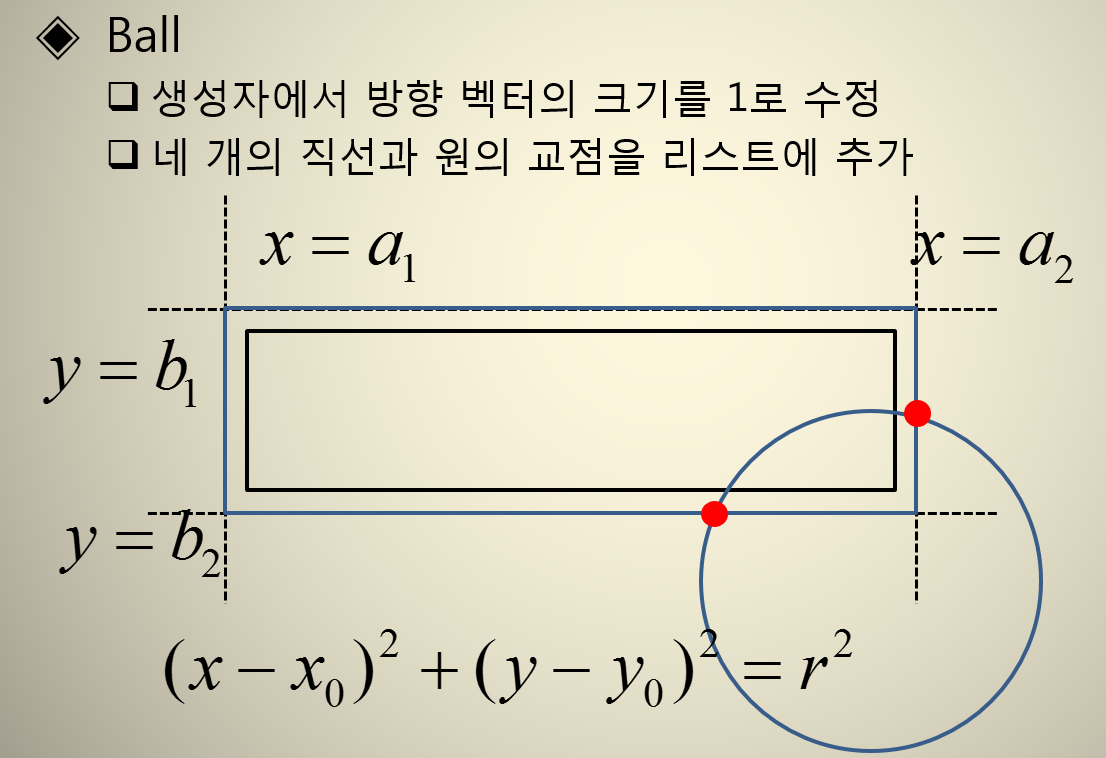
ex) Game.setup\_level(self)

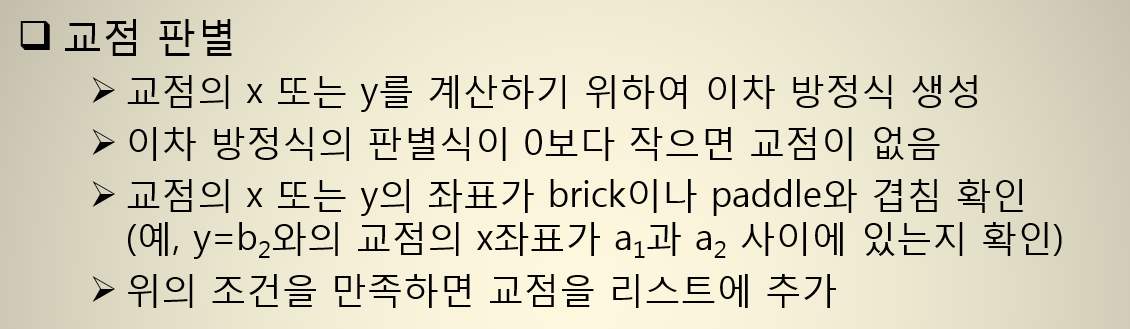
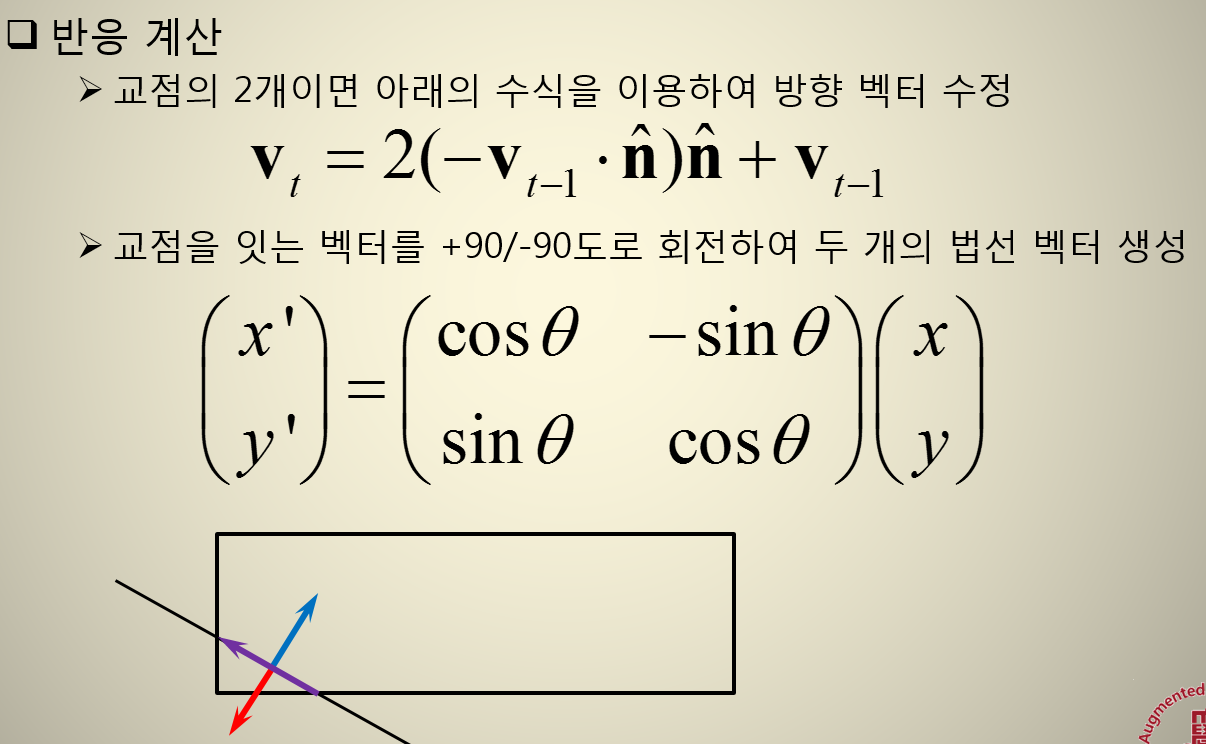
. Game.level에 따른 벽돌 층수 변경 및 임의의 벽돌만 생성

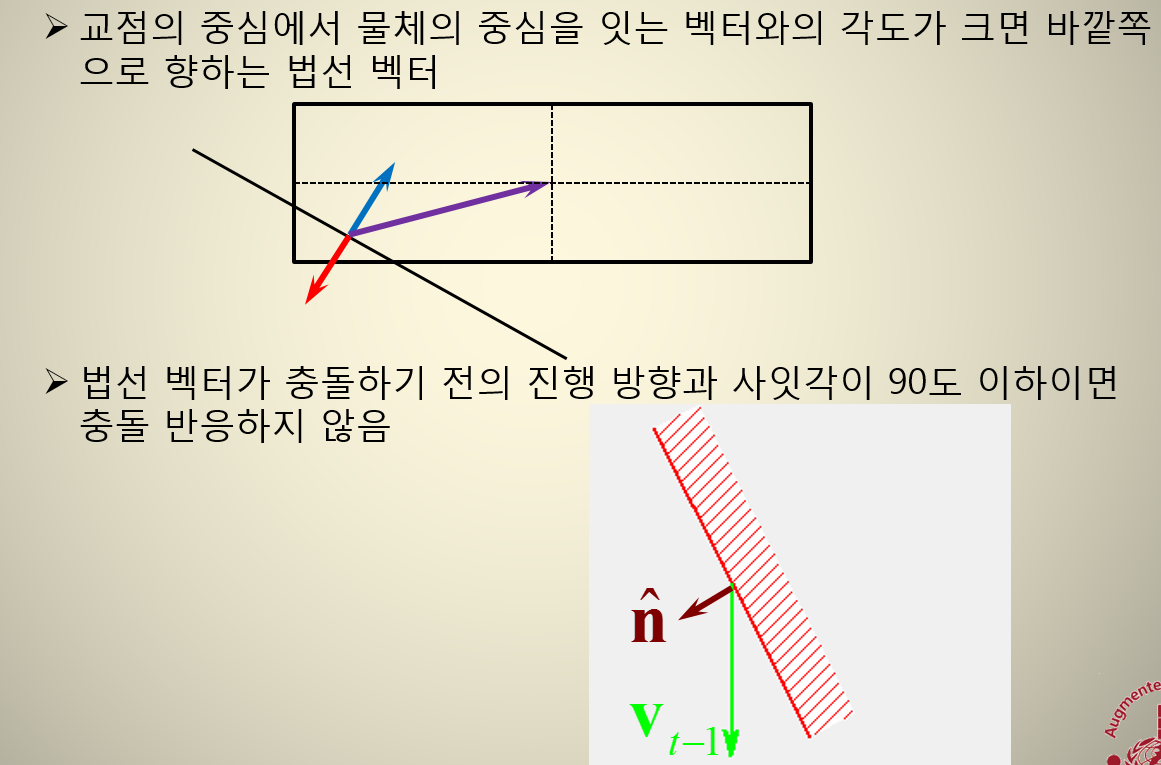
. random.randint(0, 9) 사용

|  |
| --- |
| Description]  level 5가 마지막 level, level이 증가할수록 hits와 블록 개수가 많아짐 |
| Code]  self.hits = ((1,0,0), (1,1,0), (1,1,1), (2,1,1), (2,2,1), (2,2,2))  def setup\_level(self):  for x in range(5, self.width - 5, 75):  if random.randint(0, 9) < 2self.level:  self.add\_brick(x + 37.5, 50, self.hits[self.level-1][0])  if random.randint(0, 9) < 2self.level:  self.add\_brick(x + 37.5, 70, self.hits[self.level-1][1])  if random.randint(0, 9) < 2self.level:  self.add\_brick(x + 37.5, 90, self.hits[self.level-1][2])  def setup\_game(self):  self.add\_ball()  self.update\_lives\_text()  if self.lives == 3:  self.setup\_level()  self.text = self.draw\_text(300, 200,  'Press Space to start')  self.canvas.bind('<space>', lambda \_: self.start\_game())  def add\_brick(self, x, y, hits):  if hits != 0:  brick = Brick(self.canvas, x, y, hits)  self.items[brick.item] = brick  def game\_loop(self):  self.check\_collisions()  num\_bricks = len(self.canvas.find\_withtag('brick'))  if num\_bricks == 0:  self.ball.speed = None  if self.level >= 5:  self.draw\_text(300, 200, 'You win!')  else:  self.level += 1  self.lives = 3  self.setup\_game()  def add\_brick(self, x, y, hits):  if hits != 0:  brick = Brick(self.canvas, x, y, hits)  self.items[brick.item] = brick |
| Game Shot] |

3. 모서리에 충돌한 공의 충돌 반응 처리



|  |
| --- |
| Description]  각 좌표를 vector2d클래스가 아닌 tuple, list로 표현함  동시에 두개 이상의 블럭을 맞춘 경우에는 가로, 혹은 세로로 붙어있는 두개의 블럭을 맞춘 경우만 생각함  두 블럭을 하나의 큰 블럭으로 생각해, 좌표를 계산해서 충돌을 검사함  충돌체가 하나인 경우 법선벡터를 구하여 입사각을 계산함 |
| Code]  def rotate\_vector(self, vector, radian):  cos = 0 # math.cos(radian) math.cos(pi/2)가 0이 안나옴  sin = math.sin(radian)  vx = vector[0] \* cos - vector[1] \* sin  vy = vector[0] \* sin + vector[1] \* cos  return (vx, vy)  def normalize(self, v):  magnitude = math.sqrt(v[0] \*\* 2 + v[1] \*\* 2)  return (v[0] / magnitude, v[1] / magnitude)  def inner\_product(self, v1, v2): # Only two elements  return v1[0] \* v2[0] + v1[1] \* v2[1]  def get\_reflection(self, n, v):  mat = (-2 \* (n[0] \*\* 2 - (1 / 2)), -2 \* n[0] \* n[1]), (-2 \* n[0] \* n[1], -2 \* (n[1] \*\* 2 - (1 / 2)))  result = [mat[0][0] \* v[0] + mat[0][1] \* v[1], mat[1][0] \* v[0] + mat[1][1] \* v[1]]  return result  def collide(self, game\_objects):  dis = 2  coords = self.get\_position()  x = (coords[0] + coords[2]) \* 0.5 # Ball의 중심x좌표  y = (coords[1] + coords[3]) \* 0.5  if not len(game\_objects)==0:  if len(game\_objects) > 1: # 수정이 필요함, 가로, 세로로 붙어있는 두개의 블럭인 경우만 생각함  #self.direction[1] \*= -1  coords=game\_objects[0].get\_position() # width, height, 0 ,0  for obj in game\_objects:  ocrd = obj.get\_position()  if coords[0] > ocrd[0]:  coords[0] = ocrd[0]  if coords[1] > ocrd[1]:  coords[1] = ocrd[1]  if coords[2] < ocrd[2]:  coords[2] = ocrd[2]  if coords[3] < ocrd[3]:  coords[3] = ocrd[3]  else:  game\_object = game\_objects[0]  coords = game\_object.get\_position()   bcenter = (x, y)  collide\_coords = (coords[0] - dis, coords[1] - dis, coords[2] + dis, coords[3] + dis)  collision\_points = []  for i in range(4):  d = self.radius \*\* 2 - (  collide\_coords[i] - bcenter[i % 2]) \*\* 2 # bcenter[0] if i%2==0 else bcenter[1]))\*\*2  if d < 0:  continue  p1 = bcenter[1 - i % 2] - math.sqrt(d) # (bcenter[1] if i%2==0 else bcenter[0]) - math.sqrt(d)  p2 = bcenter[1 - i % 2] + math.sqrt(d) # (bcenter[1] if i%2==0 else bcenter[0]) + math.sqrt(d)  if i == 0 or i == 2: # left or right then p are y pos  if p1 > collide\_coords[1] and p1 < collide\_coords[3]:  collision\_points.append((collide\_coords[i], p1))  if p2 > collide\_coords[1] and p2 < collide\_coords[3]:  collision\_points.append((collide\_coords[i], p2))  else: # top or bottom then p abre x pos  if p1 > collide\_coords[0] and p1 < collide\_coords[2]:  collision\_points.append((p1, collide\_coords[i]))  if p2 > collide\_coords[0] and p2 < collide\_coords[2]:  collision\_points.append((p2, collide\_coords[i]))   vector = (collision\_points[0][0] - collision\_points[1][0], collision\_points[0][1] - collision\_points[1][1])  n1 = self.rotate\_vector(vector, math.pi / 2)  n2 = (-n1[0], -n1[1])  vcenter = ((coords[0] + coords[2]) / 2 - (collision\_points[0][0] + collision\_points[1][0]) / 2, \  (coords[1] + coords[3]) / 2 - (collision\_points[0][1] + collision\_points[1][1]) / 2)   if self.inner\_product(vcenter, n1) < 0:  n = n1  else:  n = n2  n = self.normalize(n)  self.direction = self.get\_reflection(n, self.direction)   for game\_object in game\_objects:  if isinstance(game\_object, Brick):  game\_object.hit() |
| Game Shot]  스크린샷, 컴퓨터, 실내, 모니터이(가) 표시된 사진  자동 생성된 설명컴퓨터, 실내, 스크린샷, 모니터이(가) 표시된 사진  자동 생성된 설명 |