1. Cocos Invaders 성능 향상

- 좌/우 벽면에서 이동에 오류를 수정

- Cannon의 laser가 연속 발사되도록 수정 (충돌 반응 포함)

|  |
| --- |
| Description]  캐논이 이동을 했을 때의 좌표가 캔버스를 넘어가는 경우 이동이 안되게 함.  Laser의 INSTANCES를 리스트로 만들어 발사 딜레이를 넘어가면 laser를 추가하고 INSTANCES에 추가해 한번에 관리할 수 있게함. |
| Code]   1. 이동 오류 수정   def update(self, elapsed):  pressed = PlayerCannon.KEYS\_PRESSED  if self.isValid:  space\_pressed = pressed[key.SPACE] == 1  self.shootdelay += elapsed  if self.shootdelay > self.shootperiod and space\_pressed:  self.shootdelay = 0  for i in range(PlayerCannon.number):  self.parent.add(PlayerShoot(self.x-30 + 2\*30\*(i+1)/(PlayerCannon.number+1), self.y + 50))  else:  if PlayerCannon.KEYS\_PRESSED[key.SPACE] == 1:  self.validate()  movement = pressed[key.RIGHT] - pressed[key.LEFT]  w = self.width \* 0.5  if movement != 0 and w <= self.x <= self.parent.width - w:  if self.x + PlayerCannon.speed[0] \* movement \* elapsed < w:  self.position = eu.Vector2(w, self.position[1])  elif self.x + PlayerCannon.speed[0] \* movement \* elapsed > self.parent.width - w:  self.position = eu.Vector2(self.parent.width - w, self.position[1])  else:  self.move(PlayerCannon.speed \* movement \* elapsed)   1. Laser 연속 발사 수정   class PlayerShoot(Shoot):  INSTANCES = []  damage = 1  speed = 400   def \_\_init\_\_(self, x, y):  super(PlayerShoot, self).\_\_init\_\_(x, y, 'img/laser.png')  self.set\_speed(PlayerShoot.speed)  self.set\_speed(self.speed \* -1)  PlayerShoot.INSTANCES.append(self)   def collide(self, other):  try:  if isinstance(other, Alien):  self.parent.update\_score(other.score)  other.kill()  self.kill()  if isinstance(other, AbilityBox):  other.kill()  self.kill()  if isinstance(other, AlienBoss):  other.on\_hit()  self.kill()  except:  pass   def on\_exit(self):  super(PlayerShoot, self).on\_exit()  PlayerShoot.INSTANCES.remove(self)  PlayerCannon.update 함수  pressed = PlayerCannon.KEYS\_PRESSED if self.isValid:  space\_pressed = pressed[key.SPACE] == 1  self.shootdelay += elapsed  if self.shootdelay > self.shootperiod and space\_pressed:  self.shootdelay = 0  for i in range(PlayerCannon.number):  self.parent.add(PlayerShoot(self.x-30 + 2\*30\*(i+1)/(PlayerCannon.number+1), self.y + 50))  GameLayer.update 함수 중  for shoot in PlayerShoot.INSTANCES:  self.collide(shoot) |
| Game Shot] |

2. 기타 수정 사항

|  |
| --- |
| Description]   1. 적을 모두 해치우면 다음 레벨로 넘어감 2. 5레벨 당 보스가 출현, 보스는 총알 패턴을 가짐, 체력이 줄어들면 패턴이 변화함 3. 레벨을 끝나면 PlayerCannon을 업그레이드할 수 있음 (보스 레벨은 업그레이드 종류가 다름) 4. Shoot을 Player가 맞았을 경우, 비활성화되어 총알을 발사할 수 없고, 충돌을 회피함, space를 누르면 다시 활성화됨 |
| Code]  import random  from enum import Enum  from collections import defaultdict  from pyglet.image import load, ImageGrid, Animation from pyglet.window import key  import cocos.layer import cocos.sprite import cocos.collision\_model as cm import cocos.euclid as eu   class GameObject(cocos.sprite.Sprite):  def \_\_init\_\_(self, image, x, y):  super(GameObject, self).\_\_init\_\_(image)  self.position = eu.Vector2(x, y)  self.cshape = cm.AARectShape(self.position,  self.width \* 0.5,  self.height \* 0.5)   def update(self, elapsed):  pass   def collide(self, other):  pass   class AbilityBox(GameObject):  # *Todo: 보스 총알 패턴* # *Todo: 이미지 수정* class TYPE(Enum):  SPEED = 1  SHOOT\_SPEED = 2  MAX\_LIVES = 3  ADDITIONAL\_SHOT = 4  DAMAGE\_UP = 5  SHOT\_LATENCY = 6   AbilityType = {  TYPE.SPEED: 'img/speed.png',  TYPE.SHOOT\_SPEED: 'img/shot\_speed.png',  TYPE.MAX\_LIVES: 'img/max\_lives.png',  TYPE.ADDITIONAL\_SHOT: 'img/additional\_shot.png',  TYPE.DAMAGE\_UP: 'img/damage\_up.png',  TYPE.SHOT\_LATENCY: 'img/shot\_latency.png'  }   CHOICE = None   def \_\_init\_\_(self, abil\_type, x, y):  super(AbilityBox, self).\_\_init\_\_(AbilityBox.AbilityType[abil\_type], x, y)  self.type = abil\_type   def on\_exit(self):  AbilityBox.CHOICE = self.type  super(AbilityBox, self).on\_exit()   class Actor(GameObject):  def move(self, offset):  self.position += offset  self.cshape.center += offset   class PlayerCannon(Actor):  KEYS\_PRESSED = defaultdict(int)  speed = eu.Vector2(200, 0)  isValid = False  number = 1   def \_\_init\_\_(self, x, y):  self.sta = ImageGrid(load("img/cannon.png"), 2, 1)  super(PlayerCannon, self).\_\_init\_\_(Animation.from\_image\_sequence(self.sta[1:], 0, loop=False), x, y)  self.shootdelay = 0.0  self.shootperiod = 1.0  self.shootpersecond = 1   def increase\_sps(self):  self.shootpersecond+=1  self.shootperiod= 1.0/self.shootpersecond   def validate(self):  PlayerCannon.isValid = True  self.image = Animation.from\_image\_sequence(self.sta[1:], 0, loop=False)  pass   def invalidate(self):  PlayerCannon.isValid = False  self.image = Animation.from\_image\_sequence(self.sta[0:], 0, loop=False)   def update(self, elapsed):  pressed = PlayerCannon.KEYS\_PRESSED  if self.isValid:  space\_pressed = pressed[key.SPACE] == 1  self.shootdelay += elapsed  if self.shootdelay > self.shootperiod and space\_pressed:  self.shootdelay = 0  for i in range(PlayerCannon.number):  self.parent.add(PlayerShoot(self.x-30 + 2\*30\*(i+1)/(PlayerCannon.number+1), self.y + 50))  else:  if PlayerCannon.KEYS\_PRESSED[key.SPACE] == 1:  self.validate()  movement = pressed[key.RIGHT] - pressed[key.LEFT]  w = self.width \* 0.5  if movement != 0 and w <= self.x <= self.parent.width - w:  if self.x + PlayerCannon.speed[0] \* movement \* elapsed < w:  self.position = eu.Vector2(w, self.position[1])  elif self.x + PlayerCannon.speed[0] \* movement \* elapsed > self.parent.width - w:  self.position = eu.Vector2(self.parent.width - w, self.position[1])  else:  self.move(PlayerCannon.speed \* movement \* elapsed)   def collide(self, other):  other.kill()   class GameLayer(cocos.layer.Layer):  is\_event\_handler = True   def on\_key\_press(self, k, \_):  PlayerCannon.KEYS\_PRESSED[k] = 1   def on\_key\_release(self, k, \_):  PlayerCannon.KEYS\_PRESSED[k] = 0   def \_\_init\_\_(self, hud):  super(GameLayer, self).\_\_init\_\_()  w, h = cocos.director.director.get\_window\_size()  self.hud = hud  self.width = w  self.height = h  self.max\_lives = 3  self.score = 0  self.level = 5  self.ability\_boxes = []  self.create\_player()  self.player.validate()  self.update\_level(self.level)  cell = 1.25 \* 50  self.collman = cm.CollisionManagerGrid(0, w, 0, h,  cell, cell)  self.schedule(self.update)   def update\_level(self, level):  self.lives = self.max\_lives  self.update\_score()  self.hud.update\_lives(self.lives)  self.hud.update\_level(self.level)  if level % 5 == 0:  self.create\_boss(level)  else:  self.create\_alien\_group(100, 300)  if level > 5:  self.alien\_group.set\_period(0.2)  else:  self.alien\_group.set\_period(1.2 - level \* 0.2) # 1 - (level -1) \* 0.2   def create\_boss(self, level):  self.boss = AlienBoss(level, 400, 500)  self.add(self.boss)   def create\_player(self):  self.player = PlayerCannon(self.width \* 0.5, 50)  self.add(self.player)   def update\_score(self, score=0):  self.score += score  self.hud.update\_score(self.score)   def select\_ability(self):  if len(self.ability\_boxes) == 0:  self.hud.show\_ability()  self.clear\_shoot()  if self.level % 5 == 0:  self.ability\_boxes = [AbilityBox(AbilityBox.TYPE.ADDITIONAL\_SHOT, 300, 400),  AbilityBox(AbilityBox.TYPE.SHOT\_LATENCY, 500, 400)]  else:  self.ability\_boxes = [AbilityBox(AbilityBox.TYPE.SPEED, 200, 400), AbilityBox(AbilityBox.TYPE.SHOOT\_SPEED, 400, 400),  AbilityBox(AbilityBox.TYPE.MAX\_LIVES, 600, 400)]   for i in self.ability\_boxes:  self.add(i)  else:  if AbilityBox.CHOICE is not None:  if AbilityBox.CHOICE is AbilityBox.TYPE.SPEED:  PlayerCannon.speed += eu.Vector2(100, 0)  elif AbilityBox.CHOICE is AbilityBox.TYPE.SHOOT\_SPEED:  PlayerShoot.speed += 50  elif AbilityBox.CHOICE is AbilityBox.TYPE.MAX\_LIVES:  self.max\_lives += 1  elif AbilityBox.CHOICE is AbilityBox.TYPE.ADDITIONAL\_SHOT:  PlayerCannon.number += 1  elif AbilityBox.CHOICE is AbilityBox.TYPE.DAMAGE\_UP:  PlayerCannon.damage += 1  elif AbilityBox.CHOICE is AbilityBox.TYPE.SHOT\_LATENCY:  self.player.increase\_sps()   for i in self.ability\_boxes:  for \_, node in self.children:  if i is node:  i.kill()  self.ability\_boxes = []  AbilityBox.CHOICE = None   def create\_alien\_group(self, x, y):  self.alien\_group = AlienGroup(x, y)  for alien in self.alien\_group:  self.add(alien)   def clear\_shoot(self):  for \_, node in self.children:  if isinstance(node, Shoot):  self.remove(node)   def update(self, dt):  # collision check  self.collman.clear()  for \_, node in self.children:  self.collman.add(node)  if not self.collman.knows(node):  self.remove(node)  for shoot in PlayerShoot.INSTANCES:  self.collide(shoot)   def new\_stage():  if len(self.ability\_boxes) == 0:  self.hud.close\_ability()  self.clear\_shoot()  self.level += 1  self.update\_level(self.level)  # when stage ended  if self.level % 5 == 0:  if self.boss.lives == 0:  self.select\_ability()  new\_stage()  #when boss stage is on  else:  if self.player.isValid and self.collide(self.player):  self.respawn\_player()  for shoot in self.boss.shoot(dt):  self.add(shoot)  self.boss.update(dt)  else:  if len(self.alien\_group) == 0: # if sum(len(column.aliens) for column in self.alien\_group.columns) == 0:  self.select\_ability()  new\_stage()  # when normal stage is on  else:  if self.player.isValid and self.collide(self.player):  self.respawn\_player()  for column in self.alien\_group.columns:  shoot = column.shoot()  if shoot is not None:  self.add(shoot)  self.alien\_group.update(dt)  if random.random() < 0.001:  self.add(MysteryShip(50, self.height - 50))   for \_, node in self.children:  node.update(dt)   def collide(self, node):  if node is not None:  for other in self.collman.iter\_colliding(node):  node.collide(other)  return True  return False   def respawn\_player(self):  self.lives -= 1  if self.lives < 0:  self.unschedule(self.update)  self.remove(self.player)  self.hud.show\_game\_over()  else:  self.hud.update\_lives(self.lives)  self.player.invalidate()   class Alien(Actor):  def load\_animation(image):  seq = ImageGrid(load(image), 2, 1)  return Animation.from\_image\_sequence(seq, 0.5)   TYPES = {  '1': (load\_animation('img/alien1.png'), 40),  '2': (load\_animation('img/alien2.png'), 20),  '3': (load\_animation('img/alien3.png'), 10)  }   @staticmethod  def from\_type(x, y, alien\_type, column):  animation, score = Alien.TYPES[alien\_type]  return Alien(animation, x, y, score, column)   def \_\_init\_\_(self, img, x, y, score, column=None):  super(Alien, self).\_\_init\_\_(img, x, y)  self.score = score  self.column = column   def collide(self):  self.kill()   def on\_exit(self):  super(Alien, self).on\_exit()  if self.column:  self.column.remove(self)   class AlienBoss(Actor):  class Direction:  def \_\_init\_\_(self, x, y):  self.x = x  self.y = y   def \_\_add\_\_(self, other):  result = AlienBoss.Direction(self.x + other.x, self.y + other.y)  return result   def \_\_sub\_\_(self, other):  result = AlienBoss.Direction(self.x - other.x, self.y - other.y)  return result   def \_\_mul\_\_(self, other):  if isinstance(other, int) or isinstance(other, float):  result = AlienBoss.Direction(self.x \* other, self.y \* other)  return result  elif isinstance(other, AlienBoss.Direction):  result = AlienBoss.Direction(self.x \* other.x, self.y \* other.y)  return result   def \_\_radd\_\_(self, other):  result = AlienBoss.Direction(self.x + other.x, self.y + other.y)  return result   def \_\_rsub\_\_(self, other):  result = AlienBoss.Direction(self.x - other.x, self.y - other.y)  return result   def \_\_rmul\_\_(self, other):  if isinstance(other, int) or isinstance(other, float):  result = AlienBoss.Direction(self.x \* other, self.y \* other)  return result  elif isinstance(other, AlienBoss.Direction):  result = AlienBoss.Direction(self.x \* other.x, self.y \* other.y)  return result   def \_\_str\_\_(self):  return "%d %d" % (self.x, self.y)   def get\_x(self):  return self.x   def get\_y(self):  return self.y    def load\_animation(image):  seq = ImageGrid(load(image), 2, 1)  return Animation.from\_image\_sequence(seq, 0.5)   TYPES = {  1: (load\_animation('img/alien\_boss1.png'), 100),  2: (load\_animation('img/alien\_boss2.png'), 200),  3: (load\_animation('img/alien\_boss3.png'), 400)  }   def \_\_init\_\_(self, level, x, y):  super(AlienBoss, self).\_\_init\_\_(AlienBoss.TYPES[(level // 5 - 1) % 3 + 1][0], x, y)  self.score = AlienBoss.TYPES[level % 3 + 1][1] \* level // 5  self.time = 0  self.phase = 0  self.max\_lives = 3 \* (level // 5)  self.lives = self.max\_lives   def on\_hit(self):  self.lives -= 1  if self.lives == 0:  self.parent.update\_score(self.score)  self.kill()   def shoot(self, elapsed):  l = AlienBoss.Direction(-1, 0)  r = AlienBoss.Direction(1, 0)  d = AlienBoss.Direction(0, -1)  u = AlienBoss.Direction(0, 1)  #print(l + 2\*d)  self.time += elapsed  # phase 1  patterns = [((1.0, (d, )),  (0.3, (l, l + 2\*d, d, r + 2\*d, r)),  (0.3, (l, l + 2\*d, d, r + 2\*d, r)),  (0.3, (l, l + 2\*d, d, r + 2\*d, r))),  # phase 2  ((1.0, (d, )),  (0.2, (l, l + 2\*d, d, r + 2\*d, r)),  (0.1, (l + 2\*d, r + 2\*d)),  (0.2, (l, l + 2\*d, d, r + 2\*d, r)),  (0.1, (l + 2\*d, r + 2\*d)),  (0.2, (l, l + 2\*d, d, r + 2\*d, r)),  (0.1, (l + 2\*d, r + 2\*d))),  # phase 3  ((1.0, (d, )),  (0.2, (l, l + 2\*d, d, r + 2\*d, r)),  (0.1, (l + 2\*d, l + 3\*d, d, r + 3\*d, r + 2\*d)),  (0.2, (l, l + 2\*d, d, r + 2\*d, r)),  (0.1, (l + 2\*d, l + 3\*d, d, r + 3\*d, r + 2\*d)),  (0.2, (l, l + 2\*d, d, r + 2\*d, r)),  (0.1, (l + 2\*d, l + 3\*d, d, r + 3\*d, r + 2\*d)))]   shoot\_list = []  delay, shoot\_dir = patterns[3 - int(3\*self.lives/self.max\_lives)][self.phase]  #print(self.max\_lives, self.lives, 3 - int(3\*self.lives/self.max\_lives))  if self.time > delay:  self.time -= delay  for dir in shoot\_dir:  shoot\_list.append(Shoot(self.x, self.y, velocity=eu.Vector2(dir.get\_x(), dir.get\_y())))  self.phase = (self.phase+1) % 4  return shoot\_list   class AlienColumn(object):  def \_\_init\_\_(self, x, y):  alien\_types = enumerate(['3', '3', '2', '2', '1'])  self.aliens = [Alien.from\_type(x, y + i \* 60, alien, self)  for i, alien in alien\_types]   def should\_turn(self, d):  if len(self.aliens) == 0:  return False  alien = self.aliens[0]  x, width = alien.x, alien.parent.width  return x >= width - 50 and d == 1 or x <= 50 and d == -1   def remove(self, alien):  self.aliens.remove(alien)   def shoot(self):  if random.random() < 0.001 and len(self.aliens) > 0:  pos = self.aliens[0].position  return Shoot(pos[0], pos[1] - 50)  return None   def \_\_len\_\_(self):  return len(self.aliens)   class AlienGroup(object):  def \_\_init\_\_(self, x, y):  self.columns = [AlienColumn(x + i \* 60, y) for i in range(10)]  self.speed = eu.Vector2(10, 0)  self.direction = 1  self.elapsed = 0.0  self.period = 1.0   def update(self, elapsed):  self.elapsed += elapsed  while self.elapsed >= self.period:  self.elapsed -= self.period  offset = self.direction \* self.speed  if self.side\_reached():  self.direction \*= -1  offset = eu.Vector2(0, -10)  for alien in self:  alien.move(offset)   def set\_period(self, period):  self.period = period   def side\_reached(self):  return any(map(lambda c: c.should\_turn(self.direction),  self.columns))   def \_\_iter\_\_(self):  for column in self.columns:  for alien in column.aliens:  yield alien   def \_\_len\_\_(self):  return sum(len(column) for column in self.columns)   class Shoot(Actor):  def \_\_init\_\_(self, x, y, img='img/shoot.png', velocity=eu.Vector2(0, -1)):  super(Shoot, self).\_\_init\_\_(img, x, y)  self.speed = 400  self.velocity = velocity  self.movement = self.speed \* self.velocity   def set\_velocity(self, v):  self.velocity = v  self.movement = self.speed \* self.velocity   def set\_speed(self, speed):  self.speed = speed  self.movement = self.speed \* self.velocity   def update(self, elapsed):  self.move(self.movement \* elapsed)   class PlayerShoot(Shoot):  INSTANCES = []  damage = 1  speed = 400   def \_\_init\_\_(self, x, y):  super(PlayerShoot, self).\_\_init\_\_(x, y, 'img/laser.png')  self.set\_speed(PlayerShoot.speed)  self.set\_speed(self.speed \* -1)  PlayerShoot.INSTANCES.append(self)   def collide(self, other):  try:  if isinstance(other, Alien):  self.parent.update\_score(other.score)  other.kill()  self.kill()  if isinstance(other, AbilityBox):  other.kill()  self.kill()  if isinstance(other, AlienBoss):  other.on\_hit()  self.kill()  except:  pass   def on\_exit(self):  super(PlayerShoot, self).on\_exit()  PlayerShoot.INSTANCES.remove(self)  class HUD(cocos.layer.Layer):  def \_\_init\_\_(self):  super(HUD, self).\_\_init\_\_()  w, h = cocos.director.director.get\_window\_size()  self.score\_text = cocos.text.Label('', font\_size=18)  self.score\_text.position = (140, h - 40)  self.lives\_text = cocos.text.Label('', font\_size=18)  self.lives\_text.position = (w - 100, h - 40)  self.level\_text = cocos.text.Label('', font\_size=18)  self.level\_text.position = (20, h - 40)  self.add(self.score\_text)  self.add(self.lives\_text)  self.add(self.level\_text)   def update\_score(self, score):  self.score\_text.element.text = 'Score: %s' % score   def update\_lives(self, lives):  self.lives\_text.element.text = 'Lives: %s' % lives   def update\_level(self, level):  self.level\_text.element.text = 'Level: %s' % level   def show\_ability(self):  self.abil\_text = cocos.text.Label('Shoot one of Abilities to Take', font\_size=18)  self.abil\_text.position = (230, 500)  self.add(self.abil\_text)   def close\_ability(self):  self.remove(self.abil\_text)   def show\_game\_over(self):  w, h = cocos.director.director.get\_window\_size()  game\_over = cocos.text.Label('Game Over', font\_size=50,  anchor\_x='center',  anchor\_y='center')  game\_over.position = w \* 0.5, h \* 0.5  self.add(game\_over)   class MysteryShip(Alien):  SCORES = [10, 50, 100, 200]   def \_\_init\_\_(self, x, y):  score = random.choice(MysteryShip.SCORES)  super(MysteryShip, self).\_\_init\_\_('img/alien4.png', x, y,  score)  self.speed = eu.Vector2(150, 0)   def update(self, elapsed):  self.move(self.speed \* elapsed)   if \_\_name\_\_ == '\_\_main\_\_':  cocos.director.director.init(caption='Cocos Invaders',  width=800, height=650)  main\_scene = cocos.scene.Scene()  hud\_layer = HUD()  main\_scene.add(hud\_layer, z=1)  game\_layer = GameLayer(hud\_layer)  main\_scene.add(game\_layer, z=0)  cocos.director.director.run(main\_scene) |
| Game Shot]  1.  2.  3.  4. |