```
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

GRID_SIZE = 20
GRID_WIDTH = SCREEN_WIDTH / GRID_SIZE
GRID_HEIGHT = SCREEN_HEIGHT / GRID_SIZE
```

게임에 사용될 화면을 생성해준다.

```
class Snake(object):
    def __init__(self):
        self.create()

    # 뱀 생성
    def create(self):
        self.length = 3
        self.positions = [(int(SCREEN_WIDTH / 2), int(SCREEN_HEIGHT / 2))]
        self.direction = random.choice([UP, DOWN, LEFT, RIGHT])

    # 뱀 방향 조정
    def control(self, xy):
        if (xy[0] * -1, xy[1] * -1) == self.direction:
            return
        else:
            self.direction = xy
```

뱀과 관련된 class를 제작한다. 먼저 뱀을 생성하기 위한 create 함수에서는 뱀의 초기 길이를 3으로 지정해준다. 또한 화면의 중앙에서 랜덤 방향으로 게임을 시작할 수 있도록 self.positions와 self.direction을 지정해준다. control함수에서는 if 문을 통해 방향을 조절해준다. xy[0] *(-1), xy[1] * (-1) == self.direction이기 위해서는 xy[0]과 xy[1]이 모두 0이어야 한다. 즉 방향의 변화가 없고 다른 경우에는 direction에 xy값을 저장해준다.

```
def move(self):
        cur = self.positions[0]
        x, y = self.direction
        new = (cur[0] + (x * GRID_SIZE)), (cur[1] + (y * GRID_SIZE))
```

move에서는 뱀이 이동할 수 있도록 한다. 머리의 위치를 cur로 받는다. 위의 control 함수에서 알 수 있듯이 self.direction에서는 뱀이 나아갈 방향이 저장되어 있다. 따라서 new에 저장된 (cur[0] + (x * GRID_SIZE)), (cur[1] + (y * GRID_SIZE))값은 원래 머리의 좌표값에서 나아갈 방향으로 grid 사이즈만큼 연산한 x와 y좌표가 저장된다. 이는 새로운 머리의 왼쪽 위 좌표값이 된다.

```
        if new in self.positions[2:]:
            sleep(1)
            self.create()
        # 뱀이 게임화면을 넘어갈 경우 뱀 처음부터 다시 생성
        elif new[0] < 0 or new[0] >= SCREEN_WIDTH or \
                new[1] < 0 or new[1] >= SCREEN_HEIGHT:
            sleep(1)
            self.create()
        # 뱀이 정상적으로 이동하는 경우
        else:
            self.positions.insert(0, new)
            if len(self.positions) > self.length:
                self.positions.pop()
```

만약 새롭게 생성된 머리의 기준 좌표가 self.position[2:]에 포함된다면 이는 뱀의 머리가 뱀의 몸에 닿았다는 말이므로 게임은 초기화되어야 한다. 만약 뱀의 머리가 화면밖을 넘어가도 게임이 초기화되어야 하므로 위의 elif문을 이용한다. 이 두가지 경우를 제외하면(장애물을 만났을 때는 check함수를 통해 따로 생각함) 뱀은 정상적으로 움직인다. 따라서 self.position의 첫번째 자리에 new좌표를 넣어주고 원래 꼬리였던 부분은 삭제된다. 뱀이 먹이를 먹지 않았을 경우를 이야기하고 있기 때문에 self.length는 변함이 없다. 이 때 insert를 통해 새로운 머리의 기준 좌표가 삽입되면 self.positons의 len값이 증가하기 때문에 if문에 의해 원래 맨 뒤에 있던 꼬리의 기준 좌표는 삭제될 수 밖에 없다.

```
    # 뱀 그리기
    def draw(self, screen, s):
        if s == 'g':
            red, green, blue = 50 / (self.length), 150, 150 / (self.length)
            for i, p in enumerate(self.positions):
                color = (100 + red * i, green, blue * i)
                rect = pygame.Rect((p[0], p[1]), (GRID_SIZE, GRID_SIZE))
                pygame.draw.rect(screen, color, rect)
        elif s == 'b':
            red, green, blue = 50 / (self.length), 150 / (self.length), 200
            for i, p in enumerate(self.positions):
                color = (red * i, green * i, blue)
                rect = pygame.Rect((p[0], p[1]), (GRID_SIZE, GRID_SIZE))
                pygame.draw.rect(screen, color, rect)
```

s는 뱀1과 뱀2의 색을 구분하기 위한 장치이다. 만약 s값이 'g'라면 뱀의 색은 초록색이 되고 'b'라면 뱀의 색은 파란색이 된다. Self.length가 길어질수록 red, blue, green(만약 뱀의 색이 초록색이라면 green값은 고정, 뱀의 색이 파란색이라면 blue값은 고정)의 값은 작아져 앞의 사각형의 색과 차이가 발생한다. for문과 Pygame.draw.rect를 통해 self.positions에 저장된 좌표를 기준으로 한 변의 길이가 grid 사이즈인 사각형이 그려진다.

```
# 먹이 객체
```

```
class Feed(object):
    def __init__(self):
        self.position = (0, 0)
        self.color = ORANGE
        self.create()

    # 먹이 생성
    def create(self):
        x = random.randint(0, GRID_WIDTH - 1)
        y = random.randint(0, GRID_HEIGHT - 1)
        self.position = x * GRID_SIZE, y * GRID_SIZE

    # 먹이 그리기
    def draw(self, screen):
        rect = pygame.Rect((self.position[0], self.position[1]), (GRID_SIZE,
GRID_SIZE))
        pygame.draw.rect(screen, self.color, rect)
```

먹이 class를 통해 먹이를 생성한다. 우선 create 함수를 통해 랜덤으로 위치를 정한다. 그리드의 넘버에 그리드의 사이즈가 곱해져야 한다. Draw 함수를 통해 랜덤으로 지정된 위치에 먹이가 그려진다.

```
# 뱀이 먹이를 먹을 때 호출
    def eat(self):
        self.length += 1
        pygame.mixer.music.load("pong.wav")
        pygame.mixer.music.play()
```

snake class에서 뱀이 먹이를 먹을 때 어떻게 해야하는지 eat 함수를 통해 지정해준다. 만약 뱀이 먹이를 먹으면 뱀의 length값은 1이 증가되고 pong.wav가 재생된다.

```
#랜덤박스
class Track(object):
    def __init__(self):
        self.position = (0, 0)
        self.color = (255,0,0)
        self.create()

    def create(self):
        x = random.randint(0, GRID_WIDTH - 1)
        y = random.randint(0, GRID_HEIGHT - 1)
        self.position = x * GRID_SIZE, y * GRID_SIZE

    def draw(self, screen):
        a, b = self.position[0], self.position[1]
```

```
        rect = pygame.Rect((a, b), (GRID_SIZE, GRID_SIZE))
        pygame.draw.rect(screen, self.color, rect)
        pygame.draw.polygon(screen, self.color, [[a, b], [a+GRID_SIZE//2, b-
GRID_SIZE//2], [a+GRID_SIZE-1, b]])
```

Track class에서는 랜덤박스를 생성한다. 먹이와 똑같은 방식으로 화면에 랜덤박스를 생성한다.

```
    #뱀이 랜덤박스 만날 때
    def track(self, num):
        if num == 0:
            self.length=self.length-1
            if len(self.positions) > self.length:
                self.positions.pop()
            pygame.mixer.music.load("die2.mp3")
            pygame.mixer.music.play()
        else:
            self.length=self.length+1
            pygame.mixer.music.load("pluz.mp3")
            pygame.mixer.music.play()
```

snake class의 track함수에는 뱀이 랜덤박스를 만나면 어떻게 행동하는지 나타낸다. 우선 num이 변수로 활용되는데 이 num은 밑에 check_track함수에서 랜덤으로 지정된다. Num은 0과 1중 한 값을 가지게 되는데 만약 num값이 0이라면 self.length는 1이 줄기 때문에 if문에 의해 원래 꼬리 사각형 하나가 삭제된다. 그리고 die2.mp3가 재생된다. 만약 num이 1이라면 self.length는 1이 증가한다. 따라서 꼬리 사각형이 하나 더 생긴다. 그리고 pluz.mp3가 재생된다.

```
# 게임 객체
class Game(object):
    def __init__(self):
        self.snake = Snake()
        self.feed = Feed()
        self.speed = 20
        self.track = Track()
        self.snake2 = Snake()
        self.Makedie = Makekdie()

    # 게임 이벤트 처리 및 조작
    def process_events(self):
        #list1 = [pygame.K_UP, pygame.K_DOWN, pygame.K_LEFT, pygame.K_RIGHT]
        #list2 = [pygame.K_w, pygame.K_s, pygame.K_a, pygame.K_d]
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
```

```
                    self.snake.control(UP)
                if event.key == pygame.K_DOWN:
                    self.snake.control(DOWN)
                if event.key == pygame.K_LEFT:
                    self.snake.control(LEFT)
                if event.key == pygame.K_RIGHT:
                    self.snake.control(RIGHT)
                if event.key == pygame.K_w:
                    self.snake2.control(UP)
                if event.key == pygame.K_s:
                    self.snake2.control(DOWN)
                if event.key == pygame.K_a:
                    self.snake2.control(LEFT)
                if event.key == pygame.K_d:
                    self.snake2.control(RIGHT)

        return False
```

Game class에서는 snake와 snake2를 각각 지정해주고 process_events를 통해 뱀의 움직임을 조절한다. 아래 키보드 사진에서 파란색 사각형으로 표시된 key는 파란색 뱀의 움직임을 제어하고 초록색 사각형으로 표시된 key는 초록색 뱀의 움직임을 제어한다.

```
    # 뱀이 먹이를 먹었는지 체크
    def check_eat(self, snake, feed):
        if snake.positions[0] == feed.position:
            snake.eat()
            feed.create()

    #뱀이 랜덤박스 만날 때 랜덤으로 채택
    def check_track(self, snake, track):
        if snake.positions[0] == track.position:
            num = random.randint(0,1)
            snake.track(num)
            track.create()
```

check_eat 함수와 check_track 함수를 통해 뱀의 머리 기준 좌표와 feed, track의 position 좌표가 같은지 확인한다. Check_eat함수에서는 만약 snake.positions[0] == feed.position일 때 snake.eat을 실행하고 feed.create()를 통해 먹이를 재생산한다. Check_track함수에서는 snake.positions[0] == track.position일 때 num을 랜덤으로 지정해주고 snake.track(num)을 실행한다. Num의 값에 따라 뱀의 꼬리가 잘릴지 늘어날지 결정된다. 이후 track.create함수를 통해 랜덤박스를 재생산한다.

```
    # 게임 정보 출력
    def draw_info(self, length, speed, screen, x, color):
        info = "Length: " + str(length) + "    " + "Speed: " +
str(round(speed, 2))
        font_path = resource_path("assets/NanumGothicCoding-Bold.ttf")
        font = pygame.font.Font(font_path, 26)
        text_obj = font.render(info, 1, color)
        text_rect = text_obj.get_rect()
        text_rect.x, text_rect.y = x, 10
        screen.blit(text_obj, text_rect)

    # 게임 프레임 처리
    def display_frame(self, screen):
        screen.fill(WHITE)
        self.draw_info(self.snake.length, self.speed, screen, 475, GREEN)
        self.draw_info(self.snake2.length, self.speed, screen, 10, BLUE)
        self.snake.draw(screen, 'g')
        self.snake2.draw(screen, 'b')
        self.feed.draw(screen)
        screen.blit(screen, (0, 0))
        self.track.draw(screen)
        self.Makedie.draw(screen)
```

draw_info 함수를 통해 뱀의 길이와 speed 정보를 화면 상단에 표시한다. Display_frame 함수를 통해 화면에 뱀과 그 밖의 이미지, 정보를 표시한다.

```python
def main():
    # 게임 초기화 및 환경 설정
    pygame.init()
    pygame.display.set_caption('Snake Game')

    screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
    clock = pygame.time.Clock()
    game = Game()

    done = False
    while not done:
        done = game.process_events()
        game.run_logic()
        game.display_frame(screen)
        pygame.display.flip()
        clock.tick(game.speed)

    pygame.quit()


if __name__ == '__main__':
    main()
```

main 함수에서 pygame.init()를 통해 화면을 초기화시켜준다. 이후 창을 닫기 전까지 게임을 계속 진행시킨다.

yejinnnnnhi/opencv: opencv_pygame (github.com)