```
SCREEN_WIDTH = 900
SCREEN_HEIGHT = 660

GRID_SIZE = 30
GRID_WIDTH = SCREEN_WIDTH / GRID_SIZE
GRID_HEIGHT = SCREEN_HEIGHT / GRID_SIZE

UP = (0, -1)
DOWN = (0, 1)
LEFT = (-1, 0)
RIGHT = (1, 0)

WHITE = (255, 255, 255)
ORANGE = (250, 150, 0)
GRAY = (100, 100, 100)
GREEN = (200,20,20)
```

Modified Snake Game

Kotova Polina
20191972

First setting all basic things for our game

Class Snake consists of setting the snake position, assets etc.; controlling the direction of the snake; moving the snake by direction and defining borders and self collision; adding block when eaten; drawing whole body of the snake

Class Feed consists of the setting of the block(food), and drawing the object

```
class Feed(object):
    def __init__(self, par_screen):
        self.position = (0, 0)
        self.image = pygame.image.load("assets/fruit.png").convert()
        self.image = pygame.transform.scale(self.image, (30,30))
        self.create()
        self.par_screen = par_screen

    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):
        self.par_screen.blit(self.image, (self.position[0], self.position[1]))
        pygame.display.flip()
```

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

    def create(self):
        self.length = 2
        self.image = pygame.image.load("assets/body.png").convert()
        self.image = pygame.transform.scale(self.image, (30,30))
        self.positions = [(int(SCREEN_WIDTH / 2), int(SCREEN_HEIGHT / 2))]
        self.direction = random.choice([UP, DOWN, LEFT, RIGHT])

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

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

        # dies if touches itself
        if new in self.positions[2:]:
            sleep(1)
            self.create()
        # dies if touches borders
        elif new[0] < 0 or new[0] >= SCREEN_WIDTH or \
                new[1] < 0 or new[1] >= SCREEN_HEIGHT:
            sleep(1)
            self.create()
        # normal move
        else:
            self.positions.insert(0, new)
            if len(self.positions) > self.length:
                self.positions.pop()
```

```
    # adding block
    def eat(self):
        self.length += 1

    def draw(self):
        for i, p in enumerate(self.positions):
            self.par_screen.blit(self.image, (p[0], p[1]))
        pygame.display.flip()
```

```python
class Game(object):
    def __init__(self):

        self.surface = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
        self.snake = Snake(self.surface)
        self.feed = Feed(self.surface)
        self.speed = 0
        pygame.mixer.init()
        self.play_background_music()

    def play_background_music(self):
        pygame.mixer.music.load('assets/bg_music.mp3')
        pygame.mixer.music.play(-1, 0)

    def render_background(self):
        bg = pygame.image.load("assets/background.jpg")
        bg = pygame.transform.scale(bg, (SCREEN_WIDTH,SCREEN_HEIGHT))
        self.surface.blit(bg, (0,0))

    def process_events(self):
        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)
                elif event.key == pygame.K_DOWN:
                    self.snake.control(DOWN)
                elif event.key == pygame.K_LEFT:
                    self.snake.control(LEFT)
                elif event.key == pygame.K_RIGHT:
                    self.snake.control(RIGHT)
        return False
```

Class Game includes all the set-up and functions for the snake and object **process_events** set the relation between controls and pressed keys on the keyboard.

run_logic shows the basic logic how functions should follow
check_eat check if snake touches the food

draw_info shows score and speed

display_frame shows the whole screen

```python
def run_logic(self):
    self.render_background()
    self.snake.move()
    self.check_eat(self.snake, self.feed)
    self.speed = (self.snake.length-1)

def check_eat(self, snake, feed):
    if snake.positions[0] == feed.position:
        snake.eat()
        feed.create()

def resource_path(self, relative_path):
    try:
        base_path = sys._MEIPASS
    except Exception:
        base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)

def draw_info(self, length, speed, screen):
    info = "Points: " + str(length-2) + "    " + "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, WHITE)
    text_rect = text_obj.get_rect()
    text_rect.x, text_rect.y = 10, 10
    screen.blit(text_obj, text_rect)

def display_frame(self, screen):
    #screen.fill(WHITE)
    self.render_background()
    self.draw_info(self.snake.length, self.speed, screen)
    self.snake.draw()
    self.feed.draw()
    pygame.display.flip()
```

```python
def main():
    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()
```

In the main function game setup takes place and animation is performed, game functions until it's true

Final game looks like this with custom background is added as well as apple and snake customized. Also sizes are changed and points/speed scale modified from 0. Background music plays while the game is going on.

Github Link