Wormy

This document is for noting the changes made to the original Wormy code.

The requirements:
"Change the code in the game so that obstacles are introduced, which the worm must not pass through. If the worm moves in the direction of an obstacle such that the next position of its head collides with the position of the obstacle, then movement in that direction should not be allowed (in such a situation, movement is only possible in the other two directions). The size of the obstacles should be at least one cell, and at most no more than 5% of the total number of cells."

First we get the number of obstacles and the obstacle coordinates.

```
# Minimum 5 obstacles - Maximum 5% of cells
number_of_obstacles = random.randint(5, int(CELL_WIDTH * CELL_HEIGHT *
0.05))
obstacles_coords = get_obstacle_coords(number_of_obstacles)
```

To get the number of obstacles we first get the total number of cells in the grid by multiplying the horizontal number of cells with the vertical number, we then multiply with 0.05 to get the maximum number.

The function get_obstacle_coords returns a touple of obstacle coords in a dictionary ex: {'x':10, 'y':15}. The number of obstacles as well as their positions is randomized every time the game is played.

```
def get_obstacle_coords(number_of_obstacles):
    obstacles = []
    for i in range(number_of_obstacles):
        x = random.randint(0, CELL_WIDTH - 1)
        y = random.randint(0, CELL_HEIGHT - 1)
        obstacles.append({'x':x, 'y':y})
    return tuple(obstacles)
```

To make sure the apple doesn't spawn on top of an obstacle we have to change the get_location function like so:

```
def get_random_safe_location(obstacle_coords):
    location = {'x': random.randint(0, CELL_WIDTH - 1), 'y': random.randint(0,
CELL_HEIGHT - 1)}
    while location in obstacle_coords:
        location = {'x': random.randint(0, CELL_WIDTH - 1), 'y':
random.randint(0, CELL_HEIGHT - 1)}
    return location
```

All we do here is go through a while loop until we get a location that does not overlap with an obstacle.

The new movement system still largely uses the same logic. We get the new worm head coordinates from the function “move_worm_head_in_direction” which uses the same logic as before just offloaded to a function. The change is that we first check if the position is valid then make changes to the state. If the position is not valid we don’t make any changes effectively making the worm face the wall until another direction is given. The rest of the movement logic remains the same.

```
new_head = move_worm_head_in_direction(worm_coords, direction)

if new_head in obstacles_coords:# check if move runs into obstacle
    pass
else:
    # check if worm has eaten an apple
    if worm_coords[HEAD]['x'] == apple['x'] and worm_coords[HEAD]['y'] ==
apple['y']:
        # don't remove worm's tail segment
        apple = get_random_safe_location(obstacles_coords)  # set a new apple
somewhere
    else:
        del worm_coords[-1]  # remove worm's tail segment

    worm_coords.insert(0, new_head)
```

The final change made to the code was to add a draw call for the obstacles (order does not matter in this case):

```
DISPLAY_SURF.fill(BG_COLOR)
draw_grid()
draw_worm(worm_coords)
draw_obstacles(obstacles_coords)
draw_apple(apple)
draw_score(len(worm_coords) - 3)
pygame.display.update()
CLOCK.tick(FPS)
```

```
def draw_obstacles(obstacles):
    for coord in obstacles:
        x = coord['x'] * CELL_SIZE
        y = coord['y'] * CELL_SIZE
        obstacle_rect = pygame.Rect(x, y, CELL_SIZE, CELL_SIZE)
        pygame.draw.rect(DISPLAY_SURF, OBS_COL, obstacle_rect)
        obstacle_inner_segment_rect = pygame.Rect(x + 2, y + 2, CELL_SIZE - 4,
CELL_SIZE - 4)
        pygame.draw.rect(DISPLAY_SURF, DARKGRAY, obstacle_inner_segment_rect)
```

The function goes through all of the obstacle coordinates and then draws them with the same logic as the worm with an inner and outer rectangle.

Filip Nastovski
221550 - Seis