Project 3: Documentation

Project structure

- **/graphics/...** : Graphics path
  - **/hex/...** : Hex class sprites
    - **desert0.png –** desert hex type sprite
    - **mountains0-1.png** – mountain sprites
    - **plains0-2.png –** plains sprite
    - **sea0.png –** sea sprite
  - **HexMap** class sprites
    - **map.png –** map sprite
- **(module) hex_map.py** – the HexMap implementation
  - **Classes** :
    - **ImagePolygon**
      – implementation of polygon interface based on a given image and a bounding rectangle (implements hex_type_by_point)
    - **Cache** – a class for caching pygame Surfaces. Not used in the project.
    - **Hex –** implementation of an individual Hex on the HexMap
    - **HexMap –** a class representing the map of hexes
- **(module) project3.py –** main entrypoint of the project
  - **Classes:**
    - **CountDownLatch** – implementation of threading synchronizer, countdown latch
    - **ProjectHexMap –** the main class of the project

Classes

- Class **Hex**:
  -
```
class Hex:
    neigh_directions = {
        "l": (0, -1), "r": (0, 1),
        "lu": (1, -1), "ru": (1, 1),
        "ld": (2, -1), "rd": (2, 1)
    }
    inverse_direction = {
        "l": "r", "r": "l",
        "lu": "rd", "rd": "lu",
        "ld": "ru", "ru": "ld"
    }
    hex_width = 32  # px
    scale = 1
    color = (0xff, 0xff, 0xff)

    _cache_surfaces = None
    _lock = threading.Lock()
    hex_types = {
        "sea": ["sea0.png"],
        "plains": ["plains1.png", "plains2.png"],
        "mountains": ["mountains0.png", "mountains1.png"],
        "desert": ["desert0.png"]
    }

    def __init__(self, hex_type, coordinates):
        self.coordinates = (0,coordinates[1] + coordinates[0],coordinates[2] + coordinates[0])
        self.hex_type = hex_type
        self.scale = Hex.scale
        self.hex_width = Hex.hex_width
```
  - **(method) get_surface_by_hex_type** :
    -
```
@staticmethod
def get_surface_for_hex_type(hex_type):
    if Hex._cache_surfaces is None:
        Hex._lock.acquire()
        if Hex._cache_surfaces is None:
            Hex.load_cache_surfaces()
        Hex._lock.release()
    amount = len(Hex.hex_types[hex_type])
    return Hex._cache_surfaces[hex_type + str(random.randint(0, amount - 1))].copy()

@staticmethod
```
    - Get Hex sprite by hex type (desert/plains/mountains/sea)
    - double-check locking is used (lazy init)
  - **(method) draw_arrow** :
    -
```
@staticmethod
def draw_arrow(surface, _from, _to, color):
```

- Draw an arrow pointing to a neighbouring Hex

- **(method) type_from_color** :

```
@staticmethod
def type_from_color(r, g, b):
    if g > r and g > b:
        return "plains"
    if r > b:
        return "sea"
    if b > r and b > g:
        return "desert"
    return "mountains"
```

- Determine Hex type based on the RGB value of a pixel

- **(method) bbox** :

```
@staticmethod
def bbox(hex_coords):
    min_x = min(hex_coords, key=lambda x: x[0])[0]
    max_x = max(hex_coords, key=lambda x: x[0])[0]
    min_y = min(hex_coords, key=lambda x: x[1])[1]
    max_y = max(hex_coords, key=lambda x: x[1])[1]
    return min_x, min_y, max_x, max_y
```

- Find bounding box of a set of points

- **(method) fit_surface_in_hexagon** :

```
@staticmethod
def fit_surface_in_hexagon(hex_width, hex_coords, surface: pygame.Surface):
    bbox = Hex.bbox(hex_coords)
    offset = bbox[0] - hex_width / 2, bbox[1] - hex_width / 2 + 2
    scale = max(abs(bbox[0] - bbox[2]), abs(bbox[1] - bbox[3])) / min(surface.get_width(), surface.get_height())
    return offset, pygame.transform.rotozoom(surface, 30, scale)
```

- Fit a surface (a Hex sprite) in a given Hex

- **(method) draw** :

```
def draw(self, surface: pygame.Surface, draw_image=None):
    if self.hide:
        return
    hex_coords = self.getHexCoords()
    self.offset = [surface.get_width() // 2 + self.screen_offset[0],
                   surface.get_height() // 2 + self.screen_offset[1]]
    center = self.getCenterCoordsInPx()
    center = [int(x) for x in center]
    center[0] += self.offset[0]
    center[1] += self.offset[1]
    for x in hex_coords:
        x[0] += self.offset[0]
        x[1] += self.offset[1]

    if draw_image is None or draw_image:
        s = Hex.get_surface_for_hex_type(self.hex_type)
        offset, s = Hex.fit_surface_in_hexagon(surface=s, hex_coords=hex_coords, hex_width=self.hex_width)
        surface.blit(s, offset)

    if draw_image is None or not draw_image:
        for i in range(len(hex_coords)):
            p1 = hex_coords[i]
            p2 = hex_coords[(i + 1) % len(hex_coords)]
            pygame.draw.line(surface, "black", start_pos=p1, end_pos=p2, width=2)
```

- Draw Hex object on a pygame.Surface

- **(method) getHexCoordsByCenterCoords** :

```
@staticmethod
def getHexCoordsByCenterCoords(center, width=None, scale=1):
    x, y = center
    if width is None:
        width = Hex.hex_width
    coords = 0, 2 / sqrt(3) * x + 2 * y / 3, 2 / sqrt(3) * x - 2 * y / 3
    coords = tuple(a // (2 * width * scale) for a in coords)
    return coords
```

- Get Coordinates of Hex vertices by its center coordinates

- **(method) isContainedInPolygon** :

```
def isContainedInPolygon(self, poly):
    if hasattr(poly, "can_draw_hex_at"):
        self.hide = poly.can_draw_hex_at(Point(self.getCenterCoordsInPx()))
    return poly.contains(Point(self.getCenterCoordsInPx()))
```

- Checks if a Hex is contained in a Polygon

- **(method) createNeighbour** :

```
def createNeighbour(self, hexmap, location: str, hex_type, rect):
    # if location in self.neighbours and not replace_if_exists:
    #    return self.neighbours[location], False
    coords = [x for x in self.coordinates]
    shift = Hex.neigh_directions[location]
    h = Hex.transform(coords, True)
    h[shift[0]] += shift[1]
    coords = tuple(h)
    # self_loc_for_neigh=Hex.inverse_direction[location]
    _hex = None
    _created = True
    if coords in hexmap.hex_dict:
        _hex = hexmap.getHexByCoordinates(coords)
        _created = False
    else:
        _hex = Hex(hex_type, coords)
        if not _hex.isContainedInPolygon(rect):
            return None, False
        hexmap.hex_dict[coords] = _hex
    # _hex.neighbours[self_loc_for_neigh]=self
    # self.neighbours[location]=_hex
    return _hex, _created
```

- Add a new Hex (neighbouring to another Hex) to the HexMap
- The location of the neighbour: (left/right/left upper/right upper/left bottom/right bottom)

- **(method) transform** :

```
@staticmethod
def transform(coordinates, lst=False):
    r = (0,
         coordinates[1] + coordinates[0],
         coordinates[2] + coordinates[0]
         )
    if lst:
        return list(r)
    return r
```

- Transform coordinates in the Hex basis to coordinates in 2D basis

- Class **HexMap**:

```
class HexMap:
    graphics_dir = "./graphics"

    cache_surface_name = "hexmap_hexes"
    min_zoom = 0.6
    initial_zoom = 3
    max_zoom = 5

    def __init__(self, map_size, map_poly=None, image=None):
        self._hex_list = None
        self.map_size = map_size
        self.zoom_factor = 1.1
        self._hex_w = Hex.hex_width
        self.hex_dict = dict()
        self.offset = [0, 0]
        if map_poly is None:
            self.init_map_poly(self.initial_zoom)
        else:
            self.map_poly = map_poly
        self.img = image
        self.image = None
        self._cached_surface = None
        self._op_threads = []
```

- **(method) init_map_poly** :

```
def init_map_poly(self, zoom=None):
    if zoom is None:
        zoom = self.zoom_factor
    d1 = 2.05

    x1, y1, x2, y2 = (
        -self.map_size[0] // d1,
        -self.map_size[1] // d1,
        self.map_size[0] // d1,
        self.map_size[1] // d1
    )
    self.map_poly = Polygon([
        (x1 * zoom, y1 * zoom),
        (x2 * zoom, y1 * zoom),
        (x2 * zoom, y2 * zoom),
        (x1 * zoom, y2 * zoom)
    ])
    self.offset = [x1, y1]
```

  - Init the Map polygon (location on the screen etc)
- **(method) is_preparing** :

```
def is_preparing(self):
    if len(self._op_threads) == 0:
        return False
    a = any([x.is_alive() for x in self._op_threads])
    if not a:
        self._op_threads = []
    return a
```

  - Check if any rendering threads are alive
- **(method) prepare** :

```
def prepare(self, surface):
    if self.is_preparing():
        return
    self._fillMapRectangleWithHexes()
    if self._cached_surface is None:
        self._cached_surface = pygame.Surface(
            (surface.get_width() * self.initial_zoom, surface.get_height() * self.initial_zoom))
        self._cached_surface.fill(Hex.color)
        lst = list(self.hex_dict.values())

        def f():
            for x in lst:
                x.draw(self._cached_surface)

        t = threading.Thread(target=f, daemon=True)
        self._op_threads.append(t)
        t.start()
```

- Prepare self for drawing
- fill map polygon with hexes
- initialize the _cached_surface if it is None
- initialize rendering threads

- **(method) draw** :

```
def draw(self, surface):
    if self._cached_surface is None:
        self.prepare(surface)
    s = pygame.transform.rotozoom(self._cached_surface.copy(), 0, self.zoom_factor)
    surface.blit(s, self.offset)
```

- draw the HexMap in a pygame.Surface

- **(methods) clear/zoom_in/zoom_out/move** :

```
def clear(self, clear_hexes=True):
    if self.is_preparing():
        return
    self._cached_surface = None
    if clear_hexes:
        self.hex_dict.clear()

def zoom_in(self, delta):
    if self.is_preparing():
        return
    self.zoom_factor = min(self.max_zoom, max(self.min_zoom, self.zoom_factor + delta))
    for x in self.hex_dict.values():
        x.scale = self.zoom_factor
    # Hex.hex_width=self._hex_w*self.zoom_factor
    # self.clear()

def zoom_out(self, delta):
    self.zoom_in(-delta)

def move(self, direction, delta):
    if self.is_preparing():
        return
    offset = {
```

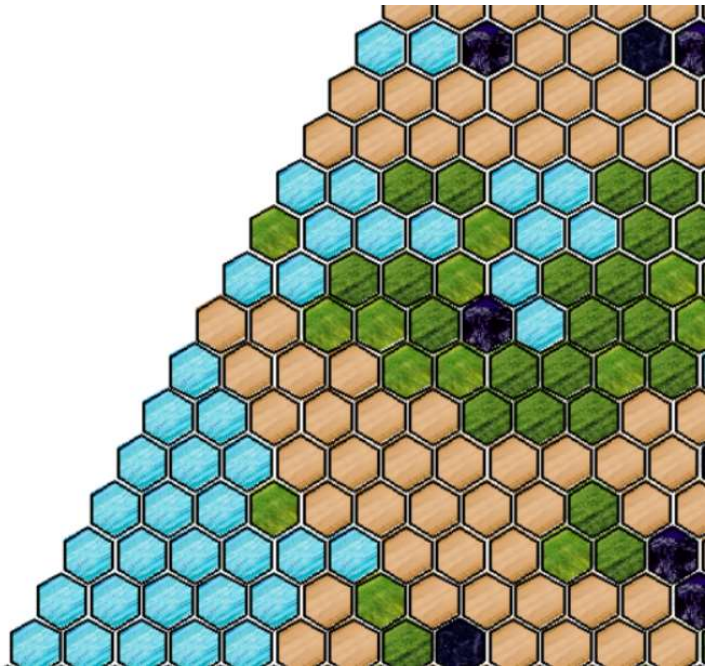
- Clear/zoom in/zoom out/move the HexMap

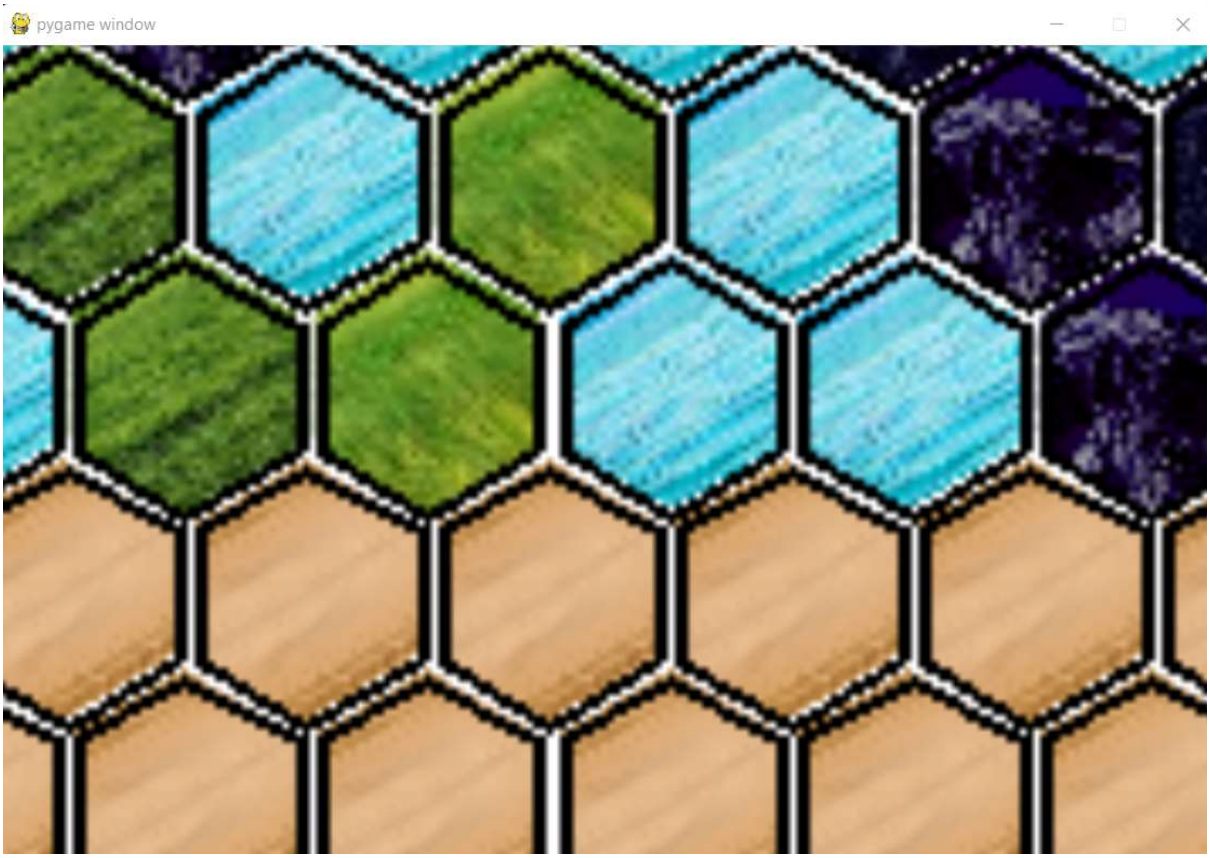
- **(method) _fillMapRectangleWithHexes** :

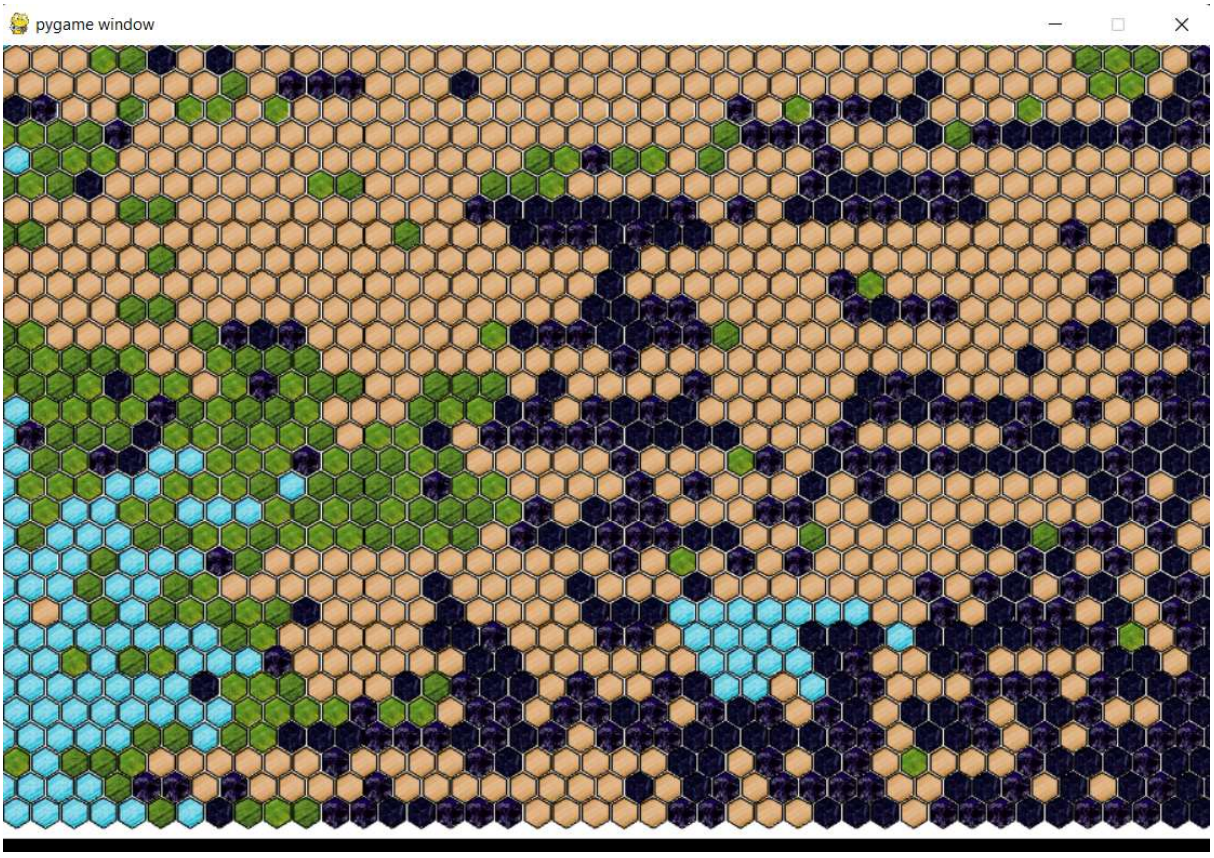
```
def _fillMapRectangleWithHexes(self):
    if len(self.hex_dict) > 0:
        return
    if hasattr(self.map_poly, "centroid"):
        p = self.map_poly.centroid
    else:
        p = Point(0, 0)
    self.init_map_poly(self.initial_zoom)
    self.image = ImagePolygon(self.img, self.map_poly)

    # Hex.hex_width=Hex.hex_width*self.max_zoom
    first_coords = Hex.getHexCoordsByCenterCoords((p.x, p.y))
    first_hex = Hex(hex_type=self.image.hex_type_by_point((p.x, p.y)), coordinates=first_coords)
    # first_hex.scale=self.max_zoom
    # first_hex.screen_offset=self.offset
    self._appendHex(first_hex)
    horizon = deque([first_hex])
    horizon_next = deque()
    hex_count_mock = 0
    hex_count = 0
    while len(horizon) > 0:
```

- Create Hex objects & build the HexMap

Project Demo








```
1
```