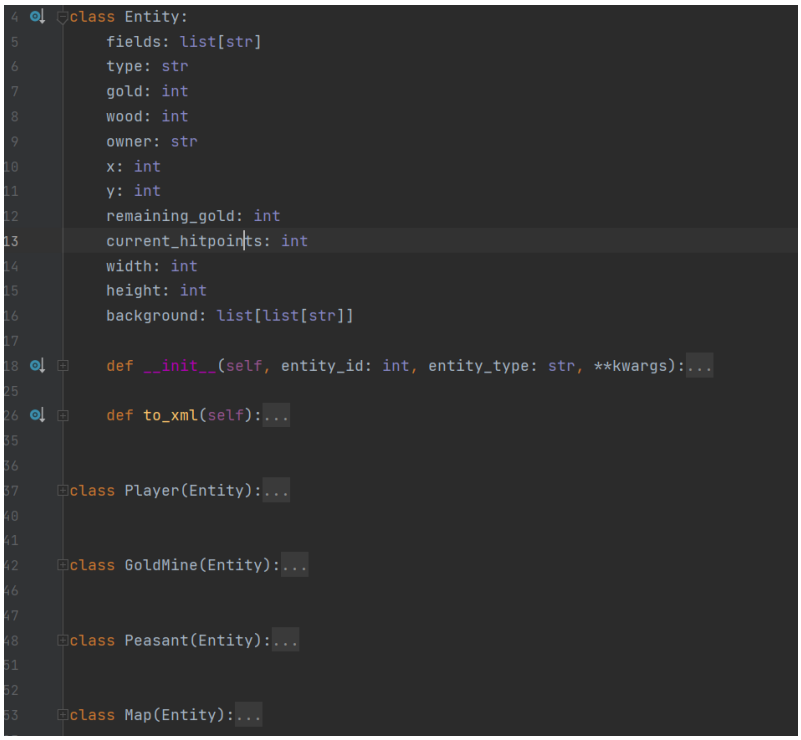
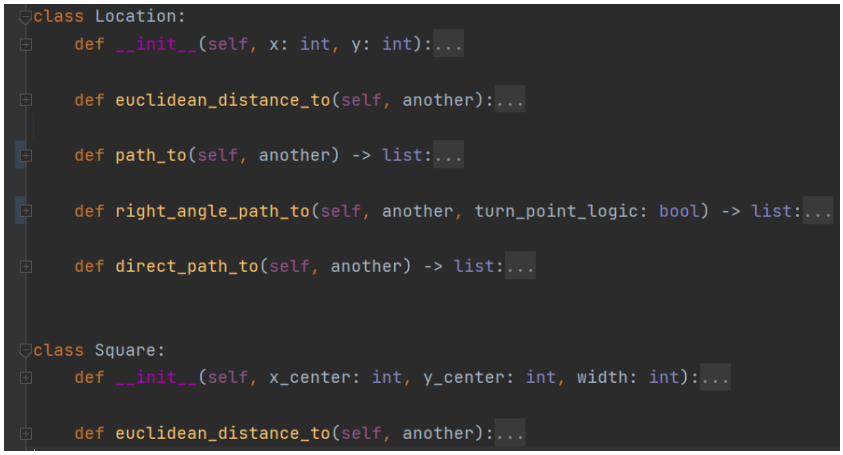
All the code resources are located inside “CS387-Project4-xf49\S3-CS387\maps”.
For this project, I mainly use **Kruskal Algorithm** for map generation. I choose Python as the programing language. To implement the map generation method, I split the whole project into three steps:
In the first step, I take a look at the structure of these built-in map files(.xml). As I understand the structure, I implement an Entity class and many subclasses of it, all of which are implemented to_xml method that we are going to use later.



Secondly, I focus on the geometry side. There are many different geometric operations we need for the Kruskal algorithm. For example, different path-finding algorithms, and distance calculation. We also need to consider the width and height of units in the game, which not allowed us to treat them simply as point locations. As you can see here, we have the Location class for point calculation, and the Square class for higher-level abstraction.



Finally, we can go to the main logic to generate the map, or we can call it GameState. The GameState class I create follows the design pattern of the factory function pattern. There are many helper pipelines in it, and each of them returns “this” so that it can support chain-style programming.

```
class GameState:
    # noinspection PyTypeChecker
    def __init__(self, width: int, height: int, base_padding="w"):...

    @staticmethod
    def random_location(center, radius_min: float, perturbation: float):...

    def can_place(self, square: Square):...

    def place_players(self):...

    def place_cross_points(self, n_cross_points: int):...

    def draw_path_to(self, l1: Location, l2: Location, padding: str, overwrite=False):...

    def add_flat_land(self):...

    def add_to_map(self, square: Square, padding: str):...

    def expand_land(self, padding: str = "l", iteration: int = 1):...

    def distribute_resource(self):...

    def to_xml(self, file_path: str = "random.xml"):...
```

And an example to use it:

```
g = GameState(60, 40).place_players().place_cross_points(6).add_flat_land() \
    .expand_land(iteration=1).expand_land("t", 1).distribute_resource()
g.to_xml()
```

Place_players can make sure the players won’t be too close to each other. I used a variant of the Kruskal algorithm in place_cross_points to make sure the whole ground is a connected graph and the ground won’t be too broad that take up all the space in the map. To prevent letting characters stuck in the road, I use expand_land to expand the roads. Also, to make the game fair for each player, I create distrubute_resource to make sure the distances from trees to each player are balanced.

If you need more detail about my project, please check my demo video:
system design and structure
directly start with demos

Wish you have a great spring break!

Thanks,
Xiao Fang