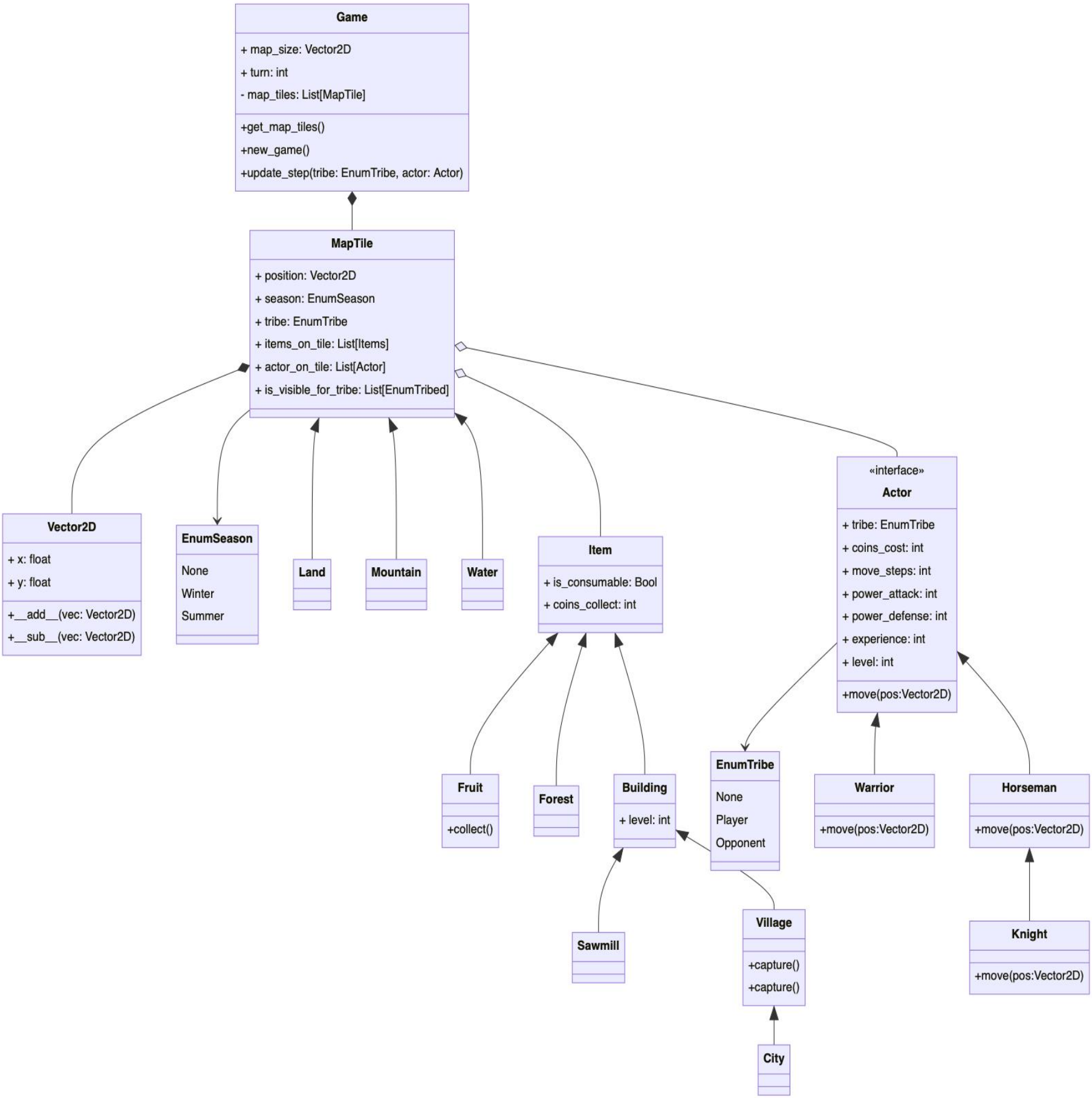
PYTHON UML CLASS DIAGRAMM



Parent Class **Game**:

```
4   class Game:
5       def __init__(self, map_size, turn, x, y, season, items, actor):
6           self.map_size = map_size
7           self.turn = turn
8           self.__map_tiles = MapTile(x, y, season, items, actor) # AGGREGATION
9
10      def get_map_tiles(self):
11          return self.__map_tiles.get_Vector2D(), self.__map_tiles.get_season(), self.__map_tiles.get_items(), self.__map_tiles.get_actor()
12
13      def new_game(self):
14          pass
15
16      def update_step(self, tribe, actor):
17          pass
```

Another parent class **MapTile**:

```
20  class MapTile:
21      def __init__(self, x, y, season, items, actor):
22          # Making an object in which we are
23          # calling the Vector2D class
24          # with proper arguments.
25          self.position = Vector2D(x, y)  # COMPOSITION
26
27          self.season = EnumSeason(season)  # no idea how to use association, so I decided to use the composition again
28          self.items_on_tile = items # AGGREGATION. Initializing the item parameter
29          self.actor_on_tile = actor
30
31      # Method which gets x and y positions
32      # with the help of get_Vector2D() method
33      # declared in the Vector2D class
34      def get_Vector2D(self):
35          return self.position.get_Vector2D()
36
37      def get_season(self):
38          return self.season.get_season()
39
40      def get_items(self):
41          return self.items_on_tile.get_items()
42
43      def get_actor(self):
44          return self.actor_on_tile.get_actor()
```

Vector2D class that is **composed with MapTile class**. Used for x/y position initialization:

```
47  class Vector2D:
48      def __init__(self, x, y):
49          self.x = x
50          self.y = y
51
52      def get_Vector2D(self):
53          return self.x, self.y
54
55      def __add__(self, other):
56          pass
57
58      def __sub__(self, other):
59          pass
```

constructor

empty func as declared

Class **EnumSeason** used in the same way as the previous **Vector2D** class but for season initialization:

```
62  class EnumSeason:
63      def __init__(self, season): # SETTING Season
64          if season == 'Winter':
65              self.season = 'Winter'
66          elif season == 'Summer':
67              self.season = 'Summer'
68          else:
69              self.season = None
70
71      def get_season(self):
72          return self.season
```

Initially, this class was meant to be implemented as Enum but I failed it. In the very last page I described it. However, **composition** works well too.

Another Parent class **Item** connected with **MapTile** class through **aggregation**:

```
87  class Item:
88      def __init__(self, is_consumable, coins_collect): # SETTING ITEM ATTRIBUTES
89          self.is_consumable = is_consumable
90          self.coins_collect = coins_collect
91
92      def get_items(self):
93          return self.is_consumable, self.coins_collect
```

How **aggregation** works:

```
182 # AGGREGATION: creating an object of the Item class in which we are passing the required parameters
183 items = Item(True, 100)
184
185 # Now we are passing the same 'items' and 'knight' objects we created earlier as a parameter to Game class
186 game = Game(10,5,100,200,'Winter', items, knight)
```

Simple **Inheritance** examples (with overriding and without):

```
96  class Fruit(Item):
97      def collect(self):
98          pass
99
100
101 class Forest(Item):
102     pass
103
104
105 class Building(Item): # Class Building inherits class Item
106     def __init__(self, is_consumable, coins_collect, level):
107         super().__init__(is_consumable, coins_collect)
108         self.level = level  # additionally overrides one  attribute
109
110     def get_items(self):
111         return super().get_items(), self.level
```

Abstract class **IActor** with abstract methods:

```python
class IActor(metaclass=ABCMeta): # interface implemented as abstract class
    @abstractmethod # with abstract methods
    def move(self):
        pass

    @abstractmethod
    def get_actor(self):
        pass
```

Child class **Actor** with attributes and overriden methods from abstract class **IActor**:

```python
class Actor(IActor):
    def __init__(self, tribe, coins_cost, move_steps, power_attack, power_defense, experience, level):
        self.tribe = EnumTribe(tribe)
        self.coins_cost = coins_cost
        self.move_steps = move_steps
        self.power_attack = power_attack
        self.power_defense = power_defense
        self.experience = experience
        self.level = level

    def move(self, x, y):
        self.x = x
        self.y = y

    def get_actor(self):
        return self.tribe.get_tribe(), self.coins_cost, self.move_steps, self.power_attack, self.power_defense, self.experience, self.level
```

Creating objects, passing arguments and calling **get_map_tiles()** method:

```python
# AGGREGATION: creating an object of the abstract class Actor (Interface) in which we are passing the required parameters
knight = Knight('Player', 10, 10, 20, 30, 0, 3)

# AGGREGATION: creating an object of the Item class in which we are passing the required parameters
items = Item(True, 100)

# Now we are passing the same 'items' and 'knight' objects we created earlier as a parameter to Game class
game = Game(10,5,100,200,'Winter', items, knight)

print(game.get_map_tiles())
```

MAP_SIZE, TURN, X, Y, SEASON

Output:

```
((100, 200), 'Winter', (True, 100), ('Player', 10, 10, 20, 30, 0, 3))

Process finished with exit code 0
```

Questions:

1. Association using Enum module was not successfull. I even couldn't get the right value making very simple example:

```python
from enum import Enum
class EnumAge(Enum):
    one = 1
    two = 2
    three = 3
class Person:
    def __init__(self, age, name):
        self.age = EnumAge
        self.name = name

    def getInfo(self):
        return self.age, self.name
misha = Person(EnumAge.one, 'Misha')
print(misha.getInfo())
```

Output:

```
(<enum 'EnumAge'>, 'Misha')

Process finished with exit code 0
```