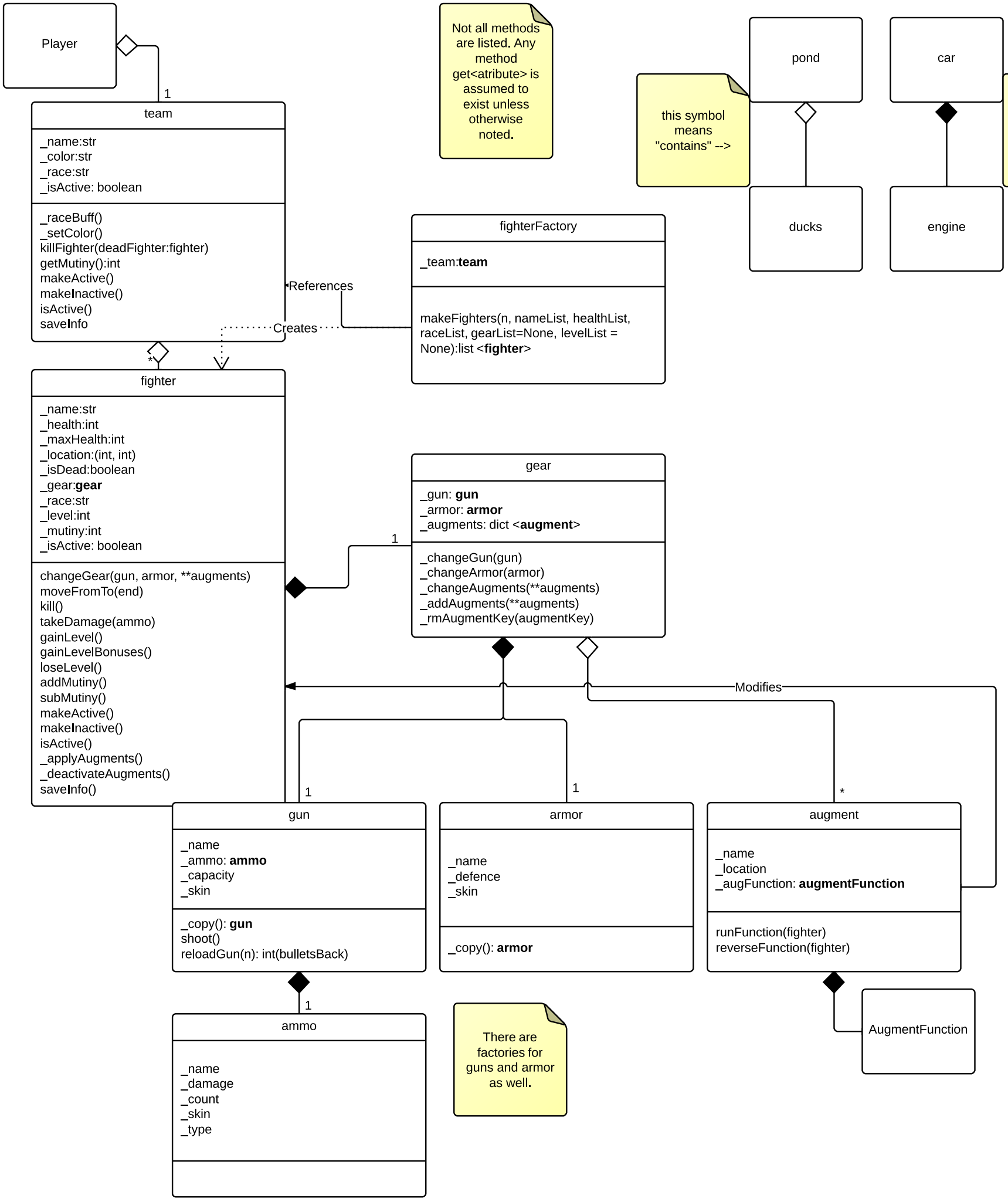

Player
1
team
_name:str
_color:str
_race:str
_isActive: boolean
_raceBuff()
_setColor()
killFighter(deadFighter:fighter)
getMutiny():int
makeActive()
makeInactive()
isActive()
saveInfo
References
Creates
*
fighterFactory
_team:team
makeFighters(n, nameList, healthList, raceList, gearList=None, levelList = None):list <fighter>
Not all methods are listed. Any method get<atribute> is assumed to exist unless otherwise noted.
pond
ducks
this symbol means "contains" -->
car
engine
this symbol means "composes" or <----"needs"
fighter
_name:str
_health:int
_maxHealth:int
_location:(int, int)
_isDead:boolean
_gear:gear
_race:str
_level:int
_mutiny:int
_isActive: boolean
changeGear(gun, armor, **augments)
moveFromTo(end)
kill()
takeDamage(ammo)
gainLevel()
gainLevelBonuses()
loseLevel()
addMutiny()
subMutiny()
makeActive()
makeInactive()
isActive()
_applyAugments()
_deactivateAugments()
saveInfo()
1
gear
_gun: gun
_armor: armor
_augments: dict <augment>
_changeGun(gun)
_changeArmor(armor)
_changeAugments(**augments)
_addAugments(**augments)
_rmAugmentKey(augmentKey)
Modifies
1
1
*
gun
_name
_ammo: ammo
_capacity
_skin
_copy(): gun
shoot()
reloadGun(n): int(bulletsBack)
armor
_name
_defence
_skin
_copy(): armor
augment
_name
_location
_augFunction: augmentFunction
runFunction(fighter)
reverseFunction(fighter)
1
ammo
_name
_damage
_count
_skin
_type
There are factories for guns and armor as well.
AugmentFunction

- <variable> denotes private! Meaning, I didn't provide a get<variable> method.

<function> is a description of the function the object adds to fighter

augmentFunction

_value
-_prevVals

effectFighter(fighter)
rmEffect(fighter)

1

augF_<function>

_value
-_prevVals

effectFighter(fighter)
rmEffect(fighter)

<---This is reimplemented in each instance

augDecorator

_value
-_decaratedAugF: **augmentFunction**
-_prevVals

effectFighter(fighter)
rmEffect(fighter)

augD_<function>

_value
-_decaratedAugF: **augmentFunction**
-_prevVals

effectFighter(fighter)
rmEffect(fighter)

<---This is reimplemented in each instance

Fish

This symbol denotes an inheritance, or an "is a" <--relationship.

Shark

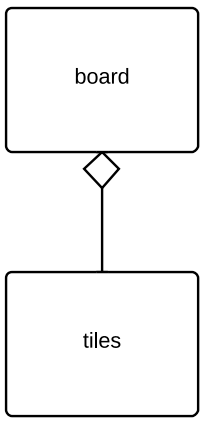
In every augD_<function>, effectFighter has the following form:

```
def effectFighter(self, fighter):
        self._dectoratedAugF.effectFighter(fighter)
        ##All the effects to the fighter go here
        ##AFTER EVERYTHING ELSE
```

and

```
def rmEffect(self, fighter):
        ##All the things that remove the effects
        ##go here BEFORE ANYTHING ELSE
        self._dectoratedAugF.rmEffect(fighter)
```

This shit isn't easy to understand.
The decorator is a kind of augmentFunction. It acts as a "wrapper" for an augmentFunction, meaning when you instantiate it, you give it an augmentFunction, and it returns a new augmentFunction that is modified. All the bold class names means they are "abstract classes." You are never supposed to instantiate them.
In our case, however, I wrote it so that "augmentFunction" can be the "base class" for any sort of augmentFunction, so you can. (This may change, but i figured it doesn't matter all that much either way.)
In this case, I kept a couple of important values in the class: _value and _prevVals. _value is meant to store a value associated with the modification. (E.G., how much health addMaxHealth adds) It could also do nothing (like in addPoisonDamage). _prevVals are values that existed on the fighter before the augment was applied (such as the kind of damage the player had before the augment was added, or the player's race before a race-change augment.)
If you still feel like you couldn't write code that uses this, text me or whatever. :DD kkkk



Stuff to come:
We are eventually going to have a drawing function for the board, fighters, etc, and we are going to have a crap-ton of accessers and methods for the tiles, one of which will be a "getTileInDirection(fromTile)" method.
The fighters/guns/etc. will eventually hold references to their drawing information in the _skin parameter, and the drawing function will have access to a dictionary mapping each _skin to a sprite generated when the game is opened.
I will write several drawing helper functions for you Ted, so you can mess around with different components of the game.