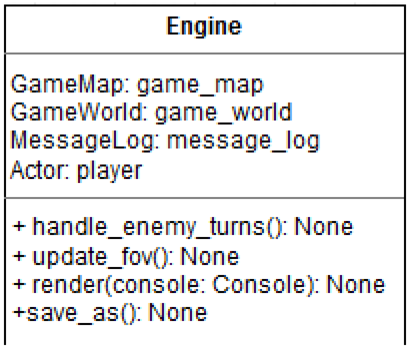
Engine: holds everything needed to render the screen and create new levels, plus a pointer to the player Actor. Also has methods for calling the ai of the enemies, updating the field of view, and rendering everything onto the console.

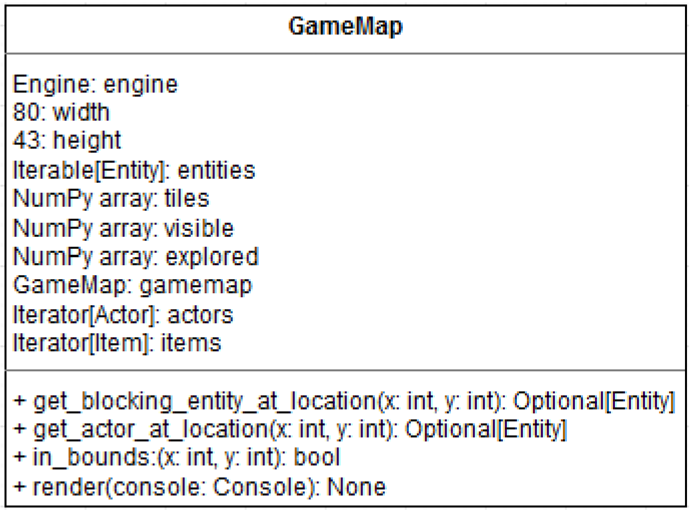


Engine

GameMap: game_map
GameWorld: game_world
MessageLog: message_log
Actor: player

+ handle_enemy_turns(): None
+ update_fov(): None
+ render(console: Console): None
+save_as(): None

GameMap: holds a 2d array of tiles, two 2d arrays to determine what tiles are visible and what have been explored, and a list of all entities. Has methods to give coordinates for specific entities, as well as the logic for rendering the map onto the console

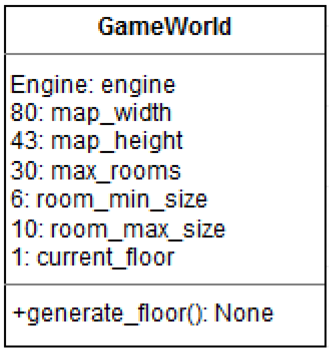


GameMap

Engine: engine
80: width
43: height
Iterable[Entity]: entities
NumPy array: tiles
NumPy array: visible
NumPy array: explored
GameMap: gamemap
Iterator[Actor]: actors
Iterator[Item]: items

+ get_blocking_entity_at_location(x: int, y: int): Optional[Entity]
+ get_actor_at_location(x: int, y: int): Optional[Entity]
+ in_bounds:(x: int, y: int): bool
+ render(console: Console): None

GameWorld: holds the parameters and methods for creating GameMaps, as well as the current floor

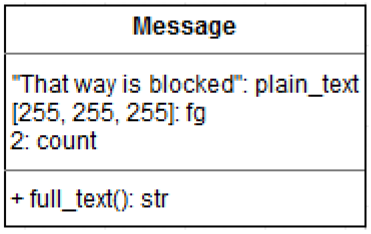


GameWorld

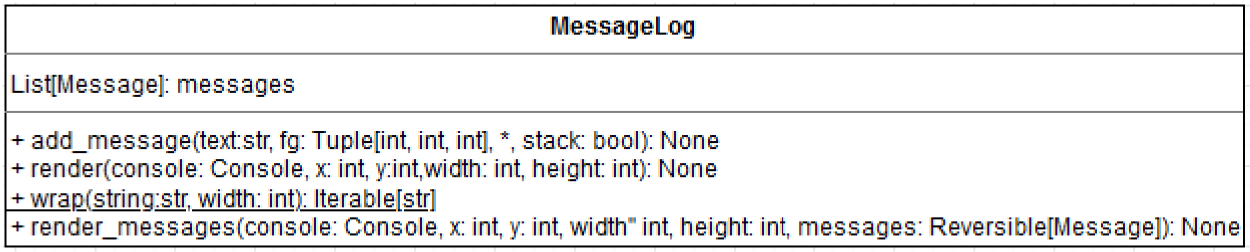
Engine: engine
80: map_width
43: map_height
30: max_rooms
6: room_min_size
10: room_max_size
1: current_floor

+generate_floor(): None

Message: holds a string, a RGB color value in a Tuple, and a count of how many times a message with the same string has been created in a row



MessageLog: holds a list of all messages that have been sent, plus a method for rendering them onto the console.

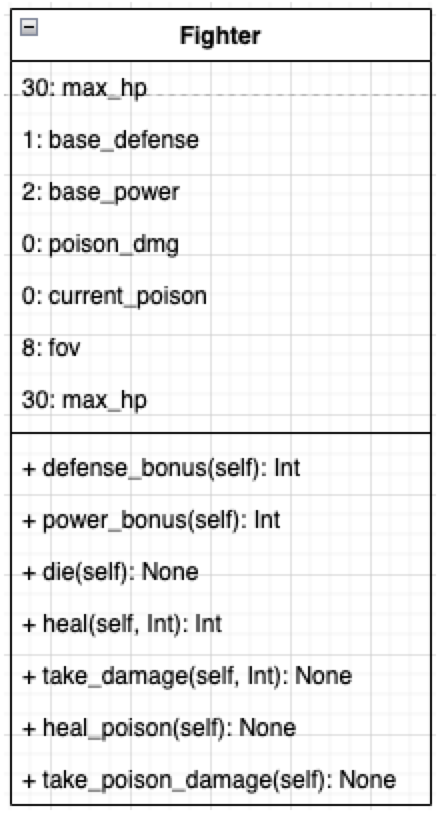

MessageLog
List[Message]: messages
+ add_message(text:str, fg: Tuple[int, int, int], *, stack: bool): None
+ render(console: Console, x: int, y:int,width: int, height: int): None
+ wrap(string:str, width: int): Iterable[str]
+ render_messages(console: Console, x: int, y: int, width" int, height: int, messages: Reversible[Message]): None

Fighter:

The purpose of the fighter class is to provide combat functionality to any Actor who should be capable of fighting. It allows them to track their hitpoints, take damage from various sources, heal damage from various sources, and die.

There are three specializations of Fighter, which represent different player character classes. Throughout the program, there are event hooks that fire functions like "fighter.use_consumable" and "fighter.on_enemy_hit". These functions on the Fighter specialization allow us to have custom class-specific abilities trigger depending on which class the player has selected. These abilities include the warrior's ability to inflict the "confused" condition with a melee attack or the mage's ability to occasionally recover consumed scrolls.

Fighter is not an abstract class, as it is still used directly by enemies in the game.


Fighter

30: max_hp

1: base_defense

2: base_power

0: poison_dmg

0: current_poison

8: fov

30: max_hp

+ defense_bonus(self): Int

+ power_bonus(self): Int

+ die(self): None

+ heal(self, Int): Int

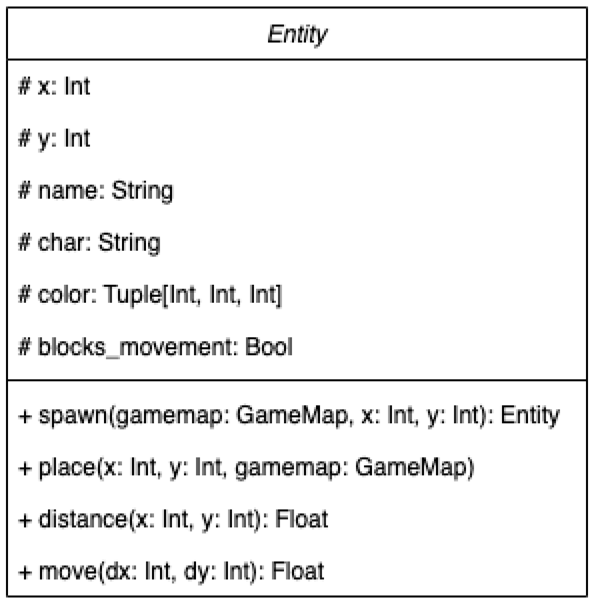
+ take_damage(self, Int): None

+ heal_poison(self): None

+ take_poison_damage(self): None

Entity:

Represents any object that can be placed in the game map. Those objects can be either movable or items.


Entity

x: Int

y: Int

name: String

char: String

color: Tuple[Int, Int, Int]

blocks_movement: Bool

+ spawn(gamemap: GameMap, x: Int, y: Int): Entity

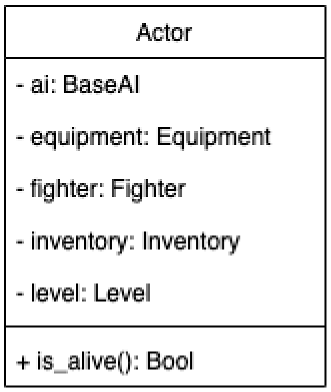
+ place(x: Int, y: Int, gamemap: GameMap)

+ distance(x: Int, y: Int): Float

+ move(dx: Int, dy: Int): Float

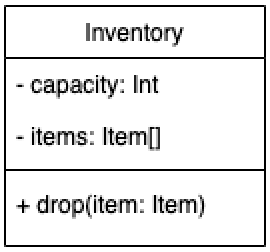
Actor:

Represents an entity that can be alive (either the player or NPCs).



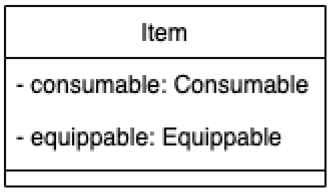
Inventory:

Contains the elements collected by an actor. Its capacity is limited.



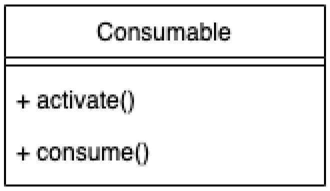
Item:

Represents an entity that can be used/activated by an actor. Can be placed in the game map.



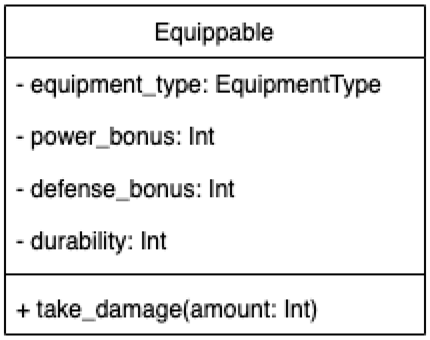
Consumable:

Represents an item that can be consumed by an actor.



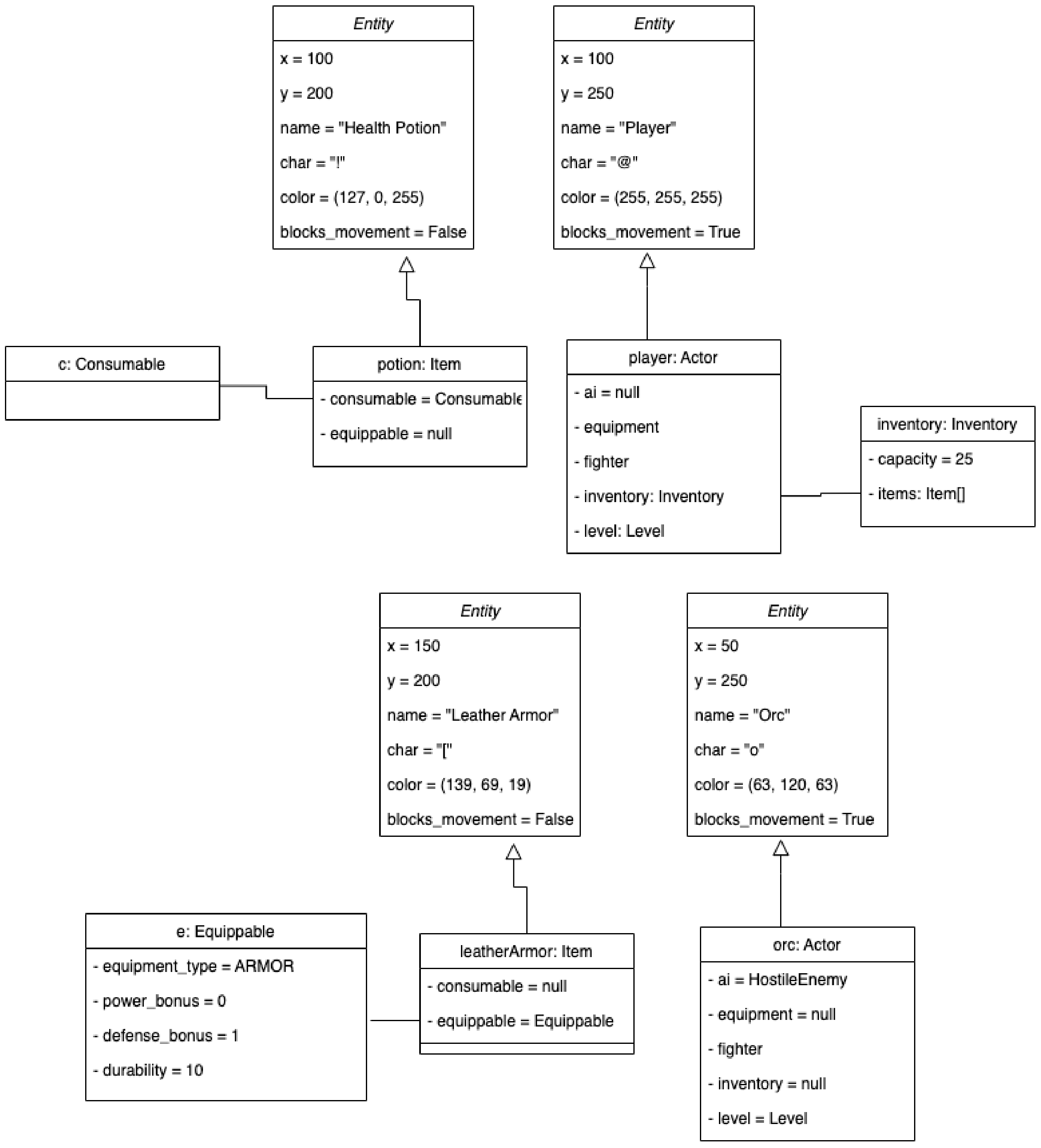
Equippable:

Represents an item that can be equipped to an actor (either armor or weapon).



Object Diagram

The following object diagram shows some player related objects, represented by the player actor and a couple items (leather armor and health potion), as well as an enemy orc:



Entity
x = 100
y = 200
name = "Health Potion"
char = "!"
color = (127, 0, 255)
blocks_movement = False
Entity
x = 100
y = 250
name = "Player"
char = "@"
color = (255, 255, 255)
blocks_movement = True
c: Consumable
potion: Item
- consumable = Consumable
- equippable = null
player: Actor
- ai = null
- equipment
- fighter
- inventory: Inventory
- level: Level
inventory: Inventory
- capacity = 25
- items: Item[]
Entity
x = 150
y = 200
name = "Leather Armor"
char = "["
color = (139, 69, 19)
blocks_movement = False
Entity
x = 50
y = 250
name = "Orc"
char = "o"
color = (63, 120, 63)
blocks_movement = True
e: Equippable
- equipment_type = ARMOR
- power_bonus = 0
- defense_bonus = 1
- durability = 10
leatherArmor: Item
- consumable = null
- equippable = Equippable
orc: Actor
- ai = HostileEnemy
- equipment = null
- fighter
- inventory = null
- level = Level