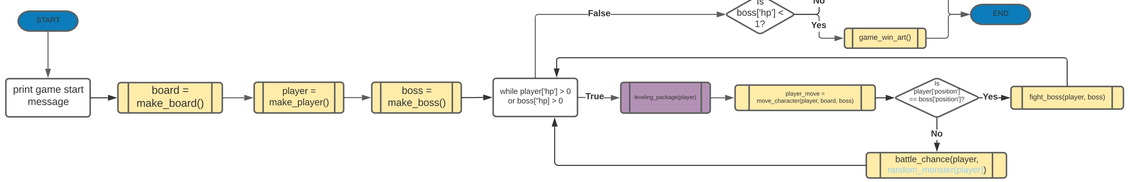
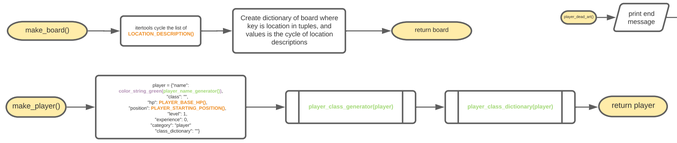
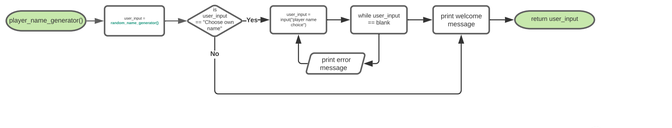
def game():

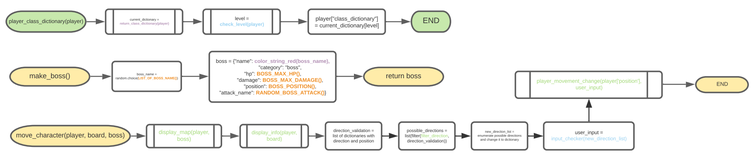

START
print game start message
board = make_board()
player = make_player()
boss = make_boss()
while player['hp'] > 0 or boss['hp'] > 0
True
leveling_package(player)
player_move = move_character(player, board, boss)
is player['position'] == boss['position']?
Yes
fight_boss(player, boss)
No
battle_chance(player, random_monster(player))
False
is boss['hp'] < 1?
No
player_dead_art()
Yes
game_win_art()
END

game_win_art()
print end message
ask user to enter anything to end the game
quit()
player_dead_art()
print end message
ask user to enter anything to end the game
quit()

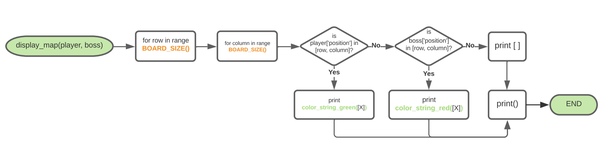

make_board()
Iterbools cycle the list of LOCATION_DESCRIPTIONS
Create dictionary of board where key is location in tuples, and values is the cycle of location descriptions
return board
make_player()
player = {"name": color_string_green(player_name_generator()), "class": "", "hp": PLAYER_BASE_HP(), "position": PLAYER_STARTING_POSITION(), "level": 1, "experience": 0, "category": "player", "class_dictionary": ""}
player_class_generator(player)
player_class_dictionary(player)
return player


player_name_generator()
user_input = random_name_generator()
is user_input == "choose own name"
Yes
No
user_input = input("player name choice")
while user_input == blank
print error message
print welcome message
return user_input
random_name_generator()
generate random name from coolname module's generate() and assign to random_name
list of choices
choices = enumerate list and change to dictionary
user_choice = input_checker(choices)
while user_choice == "I would like to choose another one at random"
False
True
user_choice == "I would like to choose this name"
No
Yes
user_choice = random_name
return user_choice
print error message

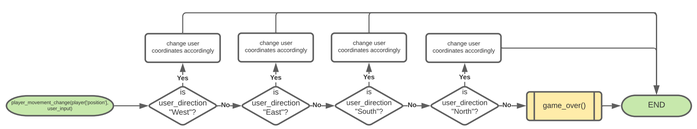
player_class_generator(player)
print message
list of available classes
choices = enumerate list and change to dictionary
user_choice = input_checker(choices)
player["class"] = user_input
print message
END


player_class_dictionary(player)
current_dictionary = class_dict_dictionary(player)
level = check_level(player)
player["class_dictionary"] = current_dictionary[level]
END
make_boss()
boss_name = random.choice(LIST_OF_BOSS_NAMES)
boss = {"name": color_string_red(boss_name), "category": "boss", "hp": BOSS_MAX_HP(), "damage": BOSS_MAX_DAMAGE(), "position": BOSS_POSITION(), "attack_name": RANDOM_BOSS_ATTACK()}
return boss

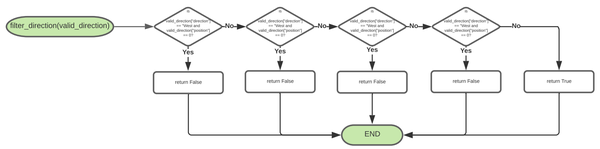
move_character(player, board, boss)
display_map(player, boss)
display_info(player, board)
direction_validation = list of directions with direction and position
possible_directions = convert_dict_to_list(direction_validation)
new_direction_list = enumerate(possible_directions) and change it to dictionary
user_input = input_checker(new_direction_list)
player_movement_change(player["position"], user_input)
END


display_map(player, boss)
for row in range BOARD_SIZE()
for column in range BOARD_SIZE()
is player["position"] in [row, column]?
Yes
No
print color_string_green([X])
is boss["position"] in [row, column]?
Yes
No
print color_string_red([X])
print [ ]
print()
END

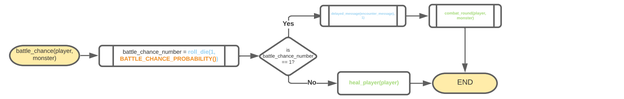
display_info(player, board)
coordinate = player["position"]
print player information by accessing keys
END


player_movement_change(player["position"], user_input)
is user_direction "West"?
is user_direction "East"?
is user_direction "South"?
is user_direction "North"?
No
Yes
change user coordinates accordingly
change user coordinates accordingly
change user coordinates accordingly
change user coordinates accordingly
game_over()
END

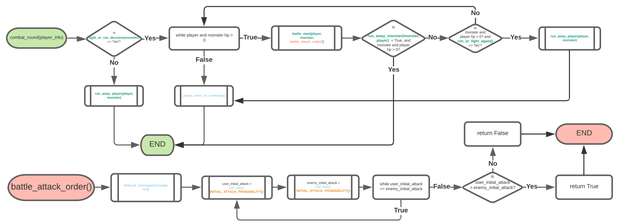
game_over()
print end message
ask user to enter anything to end the game
quit()


filter_direction(valid_direction)
valid_direction["direction"] == "West and valid_direction["position"] == 0"
valid_direction["direction"] == "West and valid_direction["position"] == 19"
valid_direction["direction"] == "East and valid_direction["position"] == 0"
valid_direction["direction"] == "East and valid_direction["position"] == 19"
Yes
No
return False
return True
END

change_dict_to_list(direction_dictionary)
possible_directions_list = []
for key_pair in directions_dictionary
append key_pair["direction"] to possible_direction_list
Yes
return possible_directions_list

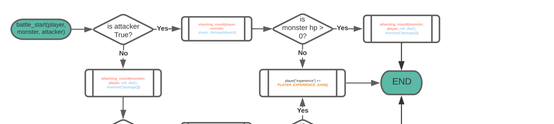

battle_chance(player, monster)
battle_chance_number = roll_die(1, BATTLE_CHANCE_PROBABILITY())
is battle_chance_number == 1?
Yes
No
battle_chance_message(battle_chance_number)
combat_round(player, monster)
heal_player(player)
END

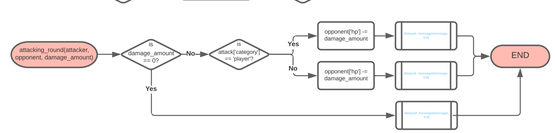
heal_player(player)
heal_message(heal_number)
health or between 0 and max_health
Yes
add PLAYER_HEAL_HP() to player['hp']
is player health greater than the max health
Yes
No
player['hp'] = player's max health
END

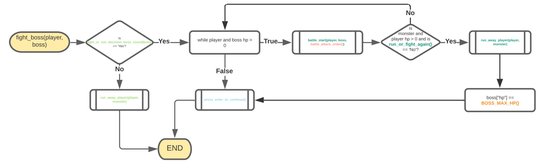

combat_round(player, info)
is fight_or_run_decision(player) == "fight"?
Yes
No
while player and monster hp > 0
True
False
battle_start(player, monster, battle_order, weapon)
is enemy_initial_attack == enemy_initial_attack and player_initial_attack == 1?
No
Yes
while user_initial_attack == enemy_initial_attack and player_initial_attack == 1
Yes
No
run_away_player(player, monster)
run_away_player(player, monster)
END
battle_attack_order()
attack_message(battle_order)
user_initial_attack = random.choice([PLAYER_ATTACK_PROBABILITY, ENEMY_ATTACK_PROBABILITY])
enemy_initial_attack = random.choice([PLAYER_ATTACK_PROBABILITY, ENEMY_ATTACK_PROBABILITY])
while user_initial_attack == enemy_initial_attack
True
False
is user_initial_attack == enemy_initial_attack?
No
Yes
return False
return True
END

fight_or_run_decision(monster)
print battle message
user_battle_decision = enumerate dictionary of YES, OR, NO()
user_choice = input_checker(user_battle_decision)
return user_choice

run_away_player(player, monster)
run_away_number = random.choice(RUN_AWAY_PROBABILITY())
is run_away_number == 1?
Yes
No
run_away_damage = random.choice(RUN_AWAY_DAMAGE_RANGE())
player['hp'] -= run_away_damage
run_away_message(run_away_number, 0)
END

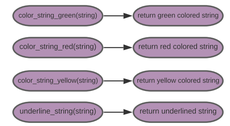

battle_start(player, monster, attacker)
is attacker True?
Yes
No
attacking_round(monster, player, monster_damage)
attacking_round(player, monster, player_damage)
is monster hp > 0?
Yes
No
attacking_round(monster, player, monster_damage)
player['experience'] += PLAYER_EXPERIENCE_GAIN()
END
is player hp > 0?
Yes
attacking_round(player, monster, player_damage)
is monster hp < 1?
Yes
No

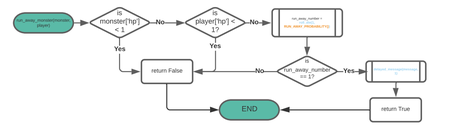

attacking_round(attacker, opponent, damage_amount)
damage_amount >= 0?
No
Yes
attack("category") == "player"?
Yes
No
opponent['hp'] -= damage_amount
opponent['hp'] -= damage_amount
attack_message(attacker, 0, 0)
attack_message(attacker, 0, 0)
attack_message(attacker, 0, 0)
END


fight_boss(player, boss)
is fight_or_run_decision_boss_number == "fight"?
Yes
No
while player and boss hp > 0
True
False
battle_start(player, boss, battle_order, weapon)
player_hp_or_run_decision = player['hp'] == 0 and is run_or_fight_again == "fight"
No
Yes
run_away_player(player, monster)
boss['hp'] += BOSS_MAX_HP()
press_enter_to_continue()
END

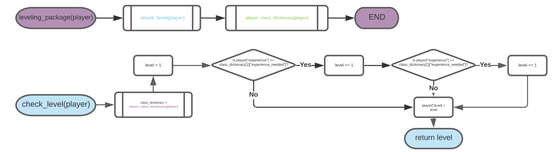
fight_or_run_decision_boss_number()
attack_message(boss, 0, 0)
user_battle_decision = enumerated dictionary of YES, OR, NO()
user_choice = input_checker(user_battle_decision)
return user_choice

Constant Functions
Player Specification
PLAYER_BASE_HP() → return 20
PLAYER_HEAL_HP() → return 4
PLAYER_STARTING_POSITION() → return [0, 0]
PLAYER_NAME_CHOICES() → return list of choosing names
PLAYER_EXPERIENCE_GAIN() → return 100
Class Specification
MAGE_HP_INCREMENT() → return 10
THIEF_HP_INCREMENT() → return 10
RANGER_HP_INCREMENT() → return 10
WARRIOR_HP_INCREMENT() → return 10
MAGE() → return dictionary of level and different specifications (e.g. damage or skill name)
THIEF() → return dictionary of level and different specifications (e.g. damage or skill name)
RANGER() → return dictionary of level and different specifications (e.g. damage or skill name)
WARRIOR() → return dictionary of level and different specifications (e.g. damage or skill name)
Boss Specification
LIST_OF_BOSS_NAME() → return a list of boss names
BOSS_MAX_HP() → return 30
BOSS_MAX_DAMAGE() → return 10
BOSS_POSITION() → return [14, 14]
RANDOM_BOSS_ATTACK() → return list of attacks
Monster Specification
BASE_MONSTER_HP() → return 10
MONSTER_HP_INCREMENT() → return 7
MONSTER_BASE_MINIMUM_DAMAGE() → return 5
MONSTER_BASE_DAMAGE() → return 10
MONSTER_DAMAGE_INCREMENT() → return 5
MONSTER_HP_AND_DAMAGE() → return dictionary of level, hp and damage difference in each level
LIST_OF_MONSTER_ATTACKS() → return list of attacks
LIST_OF_MONSTERS() → return a list of monster names
LIST_OF_MONSTER_TYPES() → return a list of monster types
Game Helper Functions
RUN_AWAY_PROBABILITY() → return 5
RUN_AWAY_DAMAGE_RANGE() → return 4
INITIAL_ATTACK_PROBABILITY() → return 100
BATTLE_CHANCE_PROBABILITY() → return 5
BOARD_SIZE() → return 25
CLASS_LIST() → return a list of class choice options
YES_OR_NO() → return a list of yes/no choice options
LOCATION_DESCRIPTION() → return a list of location description


color_string_green(string) → return green colored string
color_string_red(string) → return red colored string
color_string_yellow(string) → return yellow colored string
underline_string(string) → return underlined string


run_away_message(player)
is monster["hp"] < 1?
No
Yes
is player["hp"] < 1?
No
Yes
run_away_number = random.choice(RUN_AWAY_PROBABILITY())
return False
is run_away_number == 1?
No
Yes
attack_message(monster, 0, 0)
return True
END

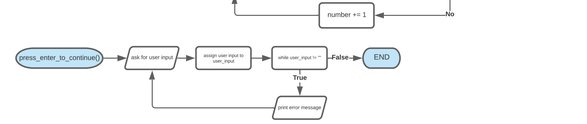
run_or_fight_again()
print battle message
user_battle_decision = enumerated dictionary of YES, OR, NO()
user_choice = input_checker(user_battle_decision)
return user_choice


leveling_package(player)
check_level(player)
assign class_dictionary(player)
END
check_level(player)
level = 1
class_dictionary = class_dictionary(player)
player['experience'] >= class_dictionary[1]["experience_needed"]
Yes
No
level += 1
player['level'] = level
return level

return_class_dictionary(player)
is player["class"] == "Mage"?
is player["class"] == "Thief"?
is player["class"] == "Ranger"?
No
Yes
return MAGE()
return THIEF()
return RANGER()
return WARRIOR()
END

random_monster(player)
random_monster_type = random.choice(LIST_OF_MONSTER_TYPES())
random_monster_name = random.choice(LIST_OF_MONSTERS_NAME())
monster = {"name": color_string_yellow(random_monster_name), "type": random_monster_type, "category": "monster", "damage": "", "attack_name": RANDOM_MONSTER_ATTACK()}
fill_monster_hp_and_damage(player, monster)
return monster

check_monster_hp_and_damage(player, monster)
player_current_level = check_level(player)
monster_hp_dictionary = MONSTER_HP_AND_DAMAGE()
monster['hp'] = monster_hp_dictionary[player_current_level]["hp"]
monster['hp'] += monster_hp_dictionary[player_current_level]["hp"]
monster['damage'] = random.randint(monster_dictionary[player_current_level]["min_damage"], monster_dictionary[player_current_level]["max_damage"])
END


input_checker(dict_of_options)
print the dict_of_options
ask for user input
assign user input to user_input
while the user_input is not a number or user_input not in dict_of_options
False
True
print error message
return dict_of_options[int(user_input)]

roll_die(number_of_rolls, number_of_sides)
total_result = 0
for number in range(number_of_rolls)
add number to total results
is it the end of the loop?
Yes
No
number += 1
return total_result
press_enter_to_continue()
ask for user input
assign user input to user_input
while user_input != ""
False
True
print error message
END

player_damage(player)
accuracy_roll = random(1, 100)
is accuracy_roll smaller than the player's accuracy rate?
Yes
No
damage = random of player's min damage and max damage
damage = 0
return damage

delayed_message(message, delay)
print message
sleep(delay)
END