W200 Summer 2020 Project 1 Reflection
Heroes of Might and Magic Monopoly board game

Announcement

I referred to the game Heroes of Might and Magic III(1999) developed by New World Computing for the names of the castles, names and stats of equipment applied in this board.

Manual

To start the game, double click the file Hero_Monopoly.py. Following the instruction to determine player numbers and name for each player. This program supports 2 ~ 6 multiple players mode. (***All the csv files should be kept in the same path as the Hero_Monopoly.py***)

Player plays as a hero moving on a Monopoly board of 36 blocks. Hero starts with 1500 gold, 1500 army. Hero moves on the board with the movement distance decided by 2 dices (1~6).

Hero can purchase or upgrade Castle on the board by gold when hero lands on it. Castles supplies heroes with gold and army after every round. When lands on the castles owned by other heroes, hero needs to pay some gold as pass fee to the castle owner otherwise can choose to try to capture the castle by commanding the army. However, hero will lose lots of gold and all his army if fails the battle. The battle result is decided by considering Hero's army size, stats of attack and defense; Castle' level, army size. For reference, equations for power of castle, hero are as below.

$$\text{Castle Power} = \frac{\text{castle.army} * 10 * \text{castle.level}}{\text{hero.defense} * 0.01}$$

$$\text{Hero Powre} = \frac{\text{hero.army} * \text{hero.attack} * 0.01}{1 + \text{castle.level} * 0.2}$$

Hero can purchase equipment with gold when arrives the market, which enhances hero's attack and defense. Hero can also sacrifice some army to increase attack and defense when arriving the arena. When hero reaches a monolith, it will teleport the hero to the next monolith.

When hero is of negative gold balance, his castles will be mortgaged automatically to pay the debt. Hero of zero gold, zero castle loses the game. The only hero who survives till the end will win the game.

Input "exit" to quit the program when the game is over. Or players may choose to exit the game in advance after every round by inputting "exit".

Reflection:

It was a challenging to develop a game especially for the very first time. At the beginning, I found no way to start for "dividing" the whole game into multiple pieces of class instances. I listed subject–object–verb relations in a normal Monopoly game. After some review, I decided to start with the most frequently show-up subject: player/Hero and the easiest and most essential verb: move. Then I added the missing part: board for this subject–object–verb relationship. Through this, I verified and employed a pattern that creating new objects class and defining the corresponding interacting methods in the hero class, when introducing new function in the game. In this way, I expanded the game with new objects as Castle, Monolith, Arena, Market, Equipment.

In this process, the debugging and test became more cumbersome with the increasing code size. Thanks to modularizing the code of class methods especially for the class Hero, I didn't need to consider too much about previous work when testing additive part. Moreover, Jupyter Notebook helped me set up for a short test more conveniently, and PyCharm helped debug more efficiently.

However, I realized that there was a lot of work for setting up and script commands when running the game for tests. I really wished to make the code more neat and cleaner. After learning some online examples for object orient projects, I added another class BoardGame to store the game setting up information and control the game procedures.

Finally, it was a difficult task to find proper criterion to decide the win-loss mode of the game. Although, the main part was referred to the Monopoly, I added the extra resource army and different incoming mechanism. I played the game myself or with my wife for hundreds of rounds to balance parameters set up and decided the final win-loss mode.

Overall, it was a struggling experience at the beginning. However, I felt some achievement and enjoyed the process when I discovered the "pattern" works for me.

During the presentation and discussion on class, I realized it was not wise way to initialize the board map, equipment list by hard typing in the code script. I modified the program now that information of board map and all categories of equipment is stored in the csv files. It makes more flexible if I extend different board maps or update new equipment.

Function List:

hero_search(name, heros):
"""Input hero's name (string), heroes list. output hero(class Hero) owns the name"""

roll_dice(dice_number):
"""Roll dices of dice_number, most 2 times in this game"""

Class List:

getoutofloop(Exception):
"""Exception class for jump out of multiple while loops"""

BoardGame:
"""Main part of the board game, store board, players list. Control game step, checking victory and delete failed player """
Attribution: heros, board, size, round.
Method:
round_check(self) """After each round of game, check if hero survive from debt, update the surviving heroes.Exit in advance between rounds if need. Check if there is a victory and exit."""
round_step(self) """Run the board game for one round. Notice of round start, all players move and inactive with space"""

Building:
"""building on the board, including castles, shop, etc"""
Attribution: name, location.

Castle(Building):
"""Building can be purchased, captured, upgraded by Hero, def hero.act_castle in Class Hero"""
Attribution: name, location, price, level, up_fee, up_ratio, owner, army, production.

Arena(Building):
"""Building where heroes can be trained, def hero.act_arena in Class Hero"""
Attribution: name, location.

Monolith(Building):
"""Building where hero is teleport to self.next_position, def hero.act_monolith in Class Hero """
Attribution: name, location, next_location.

Market(Building):
"""Building where hero can purchase equipment (Class Equipment), def

hero.act_market in Class Hero"""

Attribution: name, location, next_location, equips_stk.

Equipment:

"""Equipment stock at the store, be purchased and equipped by hero, def hero.act_market in Class Hero"""

Attribution: name, price, att_add, def_add.

Hero:

"""Player of the game is a hero. Move and interact with Castle, Market, Monolith, Arena"""

Attribution: name, location, gold, army, castle, equips, attack, defense.

Method:

move_forward(self)"""Movement decided by two dices"""

block_check(self, board) """input is hero, and board is list of all the blocks, output the type of block"""

pay_debt(self)"""when hero is of debt, sell owning castles until cover the debt or no castles left"""

act_arena(self)"""Actions heroes can take at Arena, do some training to enhance attack and defence, decided by dices"""

act_monolith(self, monolith)"""Hero teleports to monolith.next_location"""

act_market(self, market)"""Hero can purchase equipment from the market.equips_stk

act_castle(self, castle, heros)"""Actions heroes can take at a castle, input castle is the castle hero is located, heros is the list of heroes.(BoardGame.heros)"""