Marcos Valdez

valdemar

CS 162 Sec 400

Portfolio Project - Halfway Progress Report

In addition to the required RealEstateGame class, I am implementing a Player class and a Space class with two subclasses: GO and Property. The code outline is below and a description of scenario handling follows, starting on page 12.

```
class Space:
    """
    Represents a space on a game board with a unique name.
    """
    def __init__(self, name, value):
        """
        Creates a new space of specified name and value.
        :param name: string representing the space's name
        """
        pass
    def get_name(self):
        """
        Getter for name
        :return: string representing space's name
        """
        Pass
```

```
class GO(Space):
    """
    Represents a GO type of Space class object called "GO" with
    a user defined payout value. Cannot be owned and does not
    have ownership functionality.
    """
    def __init__(self, payout):
        """
        Creates a new GO space with the name "GO" and specified
        payout value.
        :param payout: int or float representing a monetary value
        """
        pass

    def get_payout(self):
        """
        Getter for payout
        :return: int or float representing space's payout value
        """
        pass
```

```
class Property(Space):
    """
    Represents a property type of Space class object with a unique name,
    a user defined rent value, and a purchase price of 5x rent. Has an
    owner that is initialized to None, can be set to a Player class object,
    and can be reverted to None.
    """
    def __init__(self, name, rent):
        """
        Creates a new Property space of specified name and rent,
        a price of 5x rent, and no owner.
        :param name: string representing the space's name
        :param rent: int or float representing a monetary value
        """
        pass

    def get_rent(self):
        """
        Getter for rent, which is the property space's value
        :return: int or float representing space's rent value
        """
        pass

    def get_price(self):
        """
        Getter for price, which is a function of the property
        space's value
        :return: int or float representing space's purchase price
        """
```

```
    pass
def get_owner(self):
    """
    Getter for owner
    :return: Player object representing space's owner (None if no owner)
    """
    pass


def set_owner(self):
    """
    Setter for owner
    :return: None
    """
    pass
```

```
class Player:
    """
    Represents a player with a name, a balance, and a position on the
    game board
    """
    def __init__(self, name, balance):
        """
        Creates a new player positioned at GO (index 0) with the specified
        name and balance.
        :param name: string representing the player's name
        :param balance: int or float representing the player's balance
        """
        pass

    def get_name(self):
        """
        Getter for name
        :return: string representing player's name
        """
        pass

    def get_balance(self):
        """
        Getter for balance
        :return: int or float representing player's balance
        """
        pass
```

```
def get_pos(self):
    """
    Getter for pos
    :return: int representing the index of the player's position
    """
    pass


def set_pos(self, pos):
    """
    Setter for pos
    :param pos: int representing the new index of player's position
    :return: None
    """
    pass


def update_balance(self, amount):
    """
    Alters player's balance by amount. Amount can be either positive
    or negative to reflect a payment or charge.
    :param amount: int or float representing a monetary value
    :return: None
    """
    pass
```

```
class RealEstateGame:
    """
    Represents a simplified version of Monopoly including the board layout,
    states of players, and current game state. Initializes to an empty board
    with no players.
    Has a method to create a list of 1 GO class object and 24 Property class
    objects (subclasses of Space) representing the game board.
    Has a method to create a new Player class object.
    Has methods to query Player objects for balance and position to track
    state of play.
    Has a method that allows a player to buy a property space, updating the
    Property object's ownership.
    Has a method that moves the player, modifying the Player object's position.
    If necessary, modifies the balance of relevant Player objects and ownership
    of relevant Space objects.
    Has a method to check for a win condition and return the winning Player
    object's name.
    """
    def __init__(self):
        """
        Creates a new game with no board and no players.
        """
        pass
```

```
def create_spaces(self, payout, rents):
    """
    Creates 1 GO class object and 24 uniquely named Property class
    objects based on the parameters and stores them in a list
    representing the game board.
    :param payout: int or float representing payout amount for GO space
    :param rents: list of 24 ints or floats representing space rent values
    :return: None
    """
    pass


def create_player(self, name, balance):
    """
    Creates a Player object based on the parameters.
    :param name: string representing player name
    :param balance: int or float representing player starting balance
    :return: None
    """
    pass


def get_player_account_balance(self, name):
    """
    Retrieves the current balance of the Player object with the passed name
    :param name: string representing player name
    :return: int or float representing player's current balance
    """
    pass
```

```
def get_player_current_position(self, name):
    """
    Retrieves the current position of the Player object with the passed name
    :param name: string representing player name
    :return: int representing the index of the player's position
            where GO is at index 0
    """
    pass


def buy_space(self, name):
    """
    Allows player to purchase a Property space. Updates Property object's
    owner data member to the Player object with the passed name if space
    is not already owned and player's balance is greater than or equal to
    space's purchase price. Reduces player's balance by amount equal to
    space's purchase price under same conditions. Does not allow purchase
    of GO space.
    :param name: string representing player name
    :return: True if purchase is successful, False otherwise
    """
    pass
```

```
def move_player(self, name, spaces):
    """
    Updates the position of Player object with the passed name. Moves the player
    to the position "spaces" indices above the previous position. Treats the board
    list as a loop in cases where this value is beyond the range of spaces. Pays
    player GO payout if player lands on or passes GO. Charges player rent if player
    lands on a space owned by a different player (paid to said player). Reduces
    player balance by rent amount or player balance (whichever is smaller) and
    increases owner player's balance by same amount. If player's account balance
    becomes 0 after rent charge, the owner of any spaces the player owns is updated
    to None. Does not move players with an existing balance of 0.
    :param name: string representing player name
    :param spaces: int in range [1..6] representing spaces to move
    :return: None
    """
    pass


def check_game_over(self):
    """
    Checks all player balances. If only 1 player has a balance greater than 0, that
    player is the winner.
    :return: string representing winning player's name (empty string if game is not over)
    """
    pass
```

```
def display(self):
    """
    Prints a depiction of the current state of the game to the console. Represents the board
    as a straight line with GO at the beginning. Shows data from each space on multiple rows
    for readability.
    Row 1: Space names
    Row 2: GO payout and property rents/prices
    Row 3: Owners
    Row 4: Player positions
    Below the board display, each player's balance is printed.
    :return: None
    """
    pass
```

DETAILED TEXT DESCRIPTIONS OF HOW TO HANDLE THE SCENARIOS

1. Determining how to store the board spaces and players

   Spaces will be stored as a list of Space class objects of subclasses GO and Property. The list will be a data member of the RealEstateGame class called _board. GO will be assigned to index 0 and all properties will follow.

   Players will be stored as a list of Player class objects. The list will be a data member of the RealEstateGame class called _players.

2. Initializing the board spaces and players

   When a RealEstateGame class object is initialized:
   - Set data member _board to empty list
   - Set data member _players to empty list

   When create_spaces(payout, rents) is called,
   Create and append to _board a GO class object with:
   - the name “GO”
   - a payout value of the parameter payout

   Loop over each rent in rents, creating and appending to _board 24 Property class objects with:
   - a unique name (enforced by the loop itself)
   - a rent value of the parameter rent
   - a purchase value of 5x the parameter rent
   - an owner of None

   When create_player(name, balance) is called, create and append to _players a Player class object with:
   - a name of the parameter name
   - a balance of the parameter balance
   - a position of 0 (the index of GO)

3. Determining how to implement player piece movement

   Player positions will be recorded as a data member of each Player class object. They will be stored as an integer representing the index of the corresponding Space object in _board.

   When move_player(name, spaces) is called,

   The balance of the Player object associated with the parameter name will be checked

   If the balance is 0, move_player will end with no further action

   Otherwise, the value of the parameter spaces will be added to the abovementioned Player object's position. If the new position is above 24, 25 will be subtracted to loop back the beginning of the board.

4. Determining how to buy board spaces, and pay and receive rents

   As mentioned in 2., ownership of a space will be recorded within data members of each Property class object.

   When buy_space(name) is called,

   The position of the Player object associated with the parameter name will be checked

   If the index is 0, the method will return False

   Otherwise, the owner of the Property object at the index equal to the Player object's position will be checked

   If the owner is not None, the method will return False

   Otherwise, the balance of the Player object will be checked against the price of the Property object

   If the price is higher than the balance, the method will return False

   Otherwise,

   The Player object's balance will be reduced by the price

   The Property object's owner will set to the Player object

   The method will return True

   (rent handling on next page)

When move_player(name, spaces) has completed the steps outlined in 3. and a player lands on a new space,

The position of the Player object associated with the parameter name will be checked

If the index is 0, move_player will end with no further action

Otherwise, the owner of the Property class object at the index will be checked

If the owner is None or the player, move_player will end with no further action

Otherwise, the player's balance will be checked against rent of the property

If the player's balance is higher, the player's balance will be reduced by the value of the property's rent and the owner's balance will be increased by the same amount

Otherwise, the player's balance will be reduced by its value (to 0) and the owner's balance will be increased by the same amount. This will initiate a loop over all properties in _board that will change the owner of each one owned by the player to None

5. Determining how to pass GO and receive the pay amount

   If, as described in 3., a player's updated position is above 24 and requires subtracting 25 to loop, the player's balance will be increased by the amount of the GO object's payout parameter.

6. Determining when the game has ended

   When check_game_over() is called, a counter will be initialized to 0 and a variable "winner" will be initialized to None.

   A loop will check the balance of each Player object in _players,

      setting winner to the name of the first player found with a balance above 0

      incrementing the counter for each balance over 0

   If the counter reaches 2, the default empty string will be returned

   Otherwise, the string in winner will be returned