Computational Thinking and Design 1D project
F02 Group 02
Koh Jia Jun
Tan Kang Min
Chen Qinyi
Emmanuel J Lopez

Documentation

ClassInit.py

Dependencies: NIL

Player Class (**name**)

Player attributes

money: Value of the player's wealth. initial value of 1500.
name: Player name, taken from the name argument passed.
properties: List of Property objects owned by player
position: Player's position on the board
otherPlayerList: a list containing all player objects excluding self, set by calling the SetPlayerList method

Player methods

SetPlayerList(allPlayerList, lostPlayerList**)**: creates otherPlayerList
- **allPlayerList** is a list of all player objects in the game
- **lostPlayerList** is a list of all the players who are bankrupt

CheckVictory(allPlayerList**):** takes in a list of all player objects and returns True/False/None as follows:
- **True**, if the player object is the last one who is not bankrupt
- **False**, if the player object is bankrupt
- **None**, if the player didn't win or lose

Trade(tradingPlayer**,** propertiesRequested**,** moneyOffered**,** propertiesOffered **,** moneyRequested**):** trades items requested for Items offered with tradingPlayer
- **tradingPlayer** is the player object to trade properties and money with
- **moneyOffered** is the money to be given from the trading player object to the player object, its defaults to 0
- **moneyRequested** is the money to be given from the player object to the trading player object, its defaults to 0
- **propertiesOffered** is a list of property objects to be transferred from the trading player object to the player object, defaults to an empty list
- **propertiesRequested** is a list of property objects to be transferred from the player object to the trading player object, defaults to an empty list

OutOfJail(roll1**,** roll2**,** bail**):** returns a Boolean for whether the player should get out of jail
- **roll1** is the roll from the first dice
- **roll2** is the roll from the second dice
- **bail** is the Boolean for whether the player plays bail or not, defaults to False

Mortgage(prop**):** prop is mortgaged and money is added to the player's balance
- **prop** is the property that needs to be mortgaged

UnMortgage(prop**):** prop is unmortgaged and the mortgage price + 10% is subtracted from the player's balance

- **prop** is the property that needs to be mortgaged

<u>Property Class (</u>*<u>owner, titleName, buyPrice, houseRent, color, numHouse</u>*<u>)</u>

Property attributes:

Owner: player object who owns the property
titleName: string of the name of the property
housedRent: tuple containing the rent prices per house
buyPrice: money required to buy the property
isMortgaged: Boolean showing if the property is mortgaged
rent: the rent for the property
color: string with the color group of the properties, used to set the price to buy a house
numHouse: number of houses on the property, ranging from 0 to 5, 5 houses makes a hotel, defaults to 0

Property methods:

BuyHouse(numToBuy): buys a house

- adds **numToBuy** to the number of houses, defaulted to 1
- increases rent according to **housedRent**

SellHouse(numToSell): sells a house

- removes **numToSell** from the number of houses, defaulted to 1
- decreases the rent according to **housedRent**

ChangeOwner(newOwner): changes the owner attribute of the property object to the owner attribute of the newOwner player object

<u>GameInit.py</u>

Dependencies: ClassInit.py, time library, pyrebase library

<u>Variables</u>

*GameInit.**user, db***: Firebase authorization keys, objects to be imported by Main.py for successful use of firebase.

*GameInit.**chance***: List of tuples, each tuple containing a chance card text at index 0, and the effect of player wealth at index 1.

*GameInit.**community_chest***: List of tuples, each tuple containing a chance card text at index 0, and the effect of player wealth at index 1.

<u>Functions</u>

*GameInit**.player_init(start_game)***: Initializes the Player objects to start the game.

Takes in

- **'H'**: Player to host online game. This initializes the online databases and creates a room for other players to join.
- **'J'**: Player to join online game. Without an existing room available, or if a game is in progress, the program will exit instead.
- **Blank string**: Game to be played locally and offline. Firebase will not be used.

It returns the following, in order:

- **player_name** is a string, or False if offline play is selected. If online play is selected, the player_name string is used as an ID.
- **player_names** is initialized as a list, collating names of every player.
- **player_list** is a list of all Player objects, which are instances of the ClassInit.Player class. Each Player object has a name attribute which corresponds to their inputted name.
- **player_number** is initialized for both off and online play, representing the number of players in the game.
- **use_firebase** is a Boolean value, determined by method of play: Local play or Online play, according to the start_game string passed. Online play returns a value of True for use_firebase.

*GameInit.**board_init()***: Initializes the Board list to start the game

No arguments are passed

It returns the following, in order:

- **board** is a list of Property objects, which are instances of the ClassInit.Property class. Each instance is given owner, titleName, buyPrice, housedRent and color attributes upon instantiation.
- **Bank** is a single Player object, an instance of the ClassInit.Player class. Bank is set to be the owner of all Property objects when board_init() is called, and starts with the attributes money = 10000000000 and name = 'Bank'.

Graphics.py

Dependencies: os library, time library

Variables

*Graphics.**main_menu***: multiline string displayed on the main menu, representing the title card of the game.
*Graphics.**first_frame*****:** multiline string representing first frame of board moving animation.
*Graphics.**second_frame*****:** multiline string representing second frame of board moving animation.
*Graphics.**third_frame*****:** multiline string representing third frame of board moving animation.
*Graphics.**fourth_frame*****:** multiline string representing fourth frame of board moving animation.
*Graphics.**win***: multiline string displayed when the player wins.
*Graphics.**lose***: multiline string displayed when the player loses.
*Graphics.**go***: multiline string displayed when the player passes GO.

Functions

*Graphics.**Board_Animation(board, initial_pos, final_pos, move_str)*****:** Prints an animation of the movement of the board.

Takes in

- **board**: List of property objects
- **initial_pos**: Integer representing initial player index
- **final_pos**: Integer representing initial player index
- **move_str**: String representing what is to be printed

It returns nothing, and prints the corresponding board movement animation as needed from the position data entered.

*Graphics.**PrintWin()***: Prints an animation upon winning the game
No arguments are passed
It returns nothing, and prints the win multiline string in a staggered fashion.

*Graphics.**PrintLose()***: Prints an animation upon losing the game
It returns nothing, and prints the lose multiline string in a staggered fashion.

TurnFunctions.py

Dependencies: Graphics.py, time library

Functions

*TurnFunctions.**count_properties(player_object, color_group_list)***: Counts the number of properties of a certain colour that a player owns.

Takes in
- **player_object**: Instance of the ClassInit.Player class.
- **color_group_list**: List of all properties in a certain colour group

It returns, in order:
- **counter**: Integer value of the number of properties from color_group_list that the player owns.
- **Boolean**: Representing whether or not the player owns every property from that colour group.

*TurnFunctions.**checkColor(property, playerProp)***: returns property if the properties passed are of the same colour.

Takes in
- **property**: Instance of the ClassInit.Property class
- **playerProp**: The instance of the ClassInit.Property class to compare to

It returns:
- **property:** if the two object's color attributes match. Else, return None.

*TurnFunctions.**evenBuild(color_group_list, currentProp, bs)***: Checks the difference between the min and maximum number of houses in the colour group

Takes in
- **color_group_list**: List of all properties in a certain colour group
- **currentProp**: The instance of the ClassInit.Property class to check

- **bs**: Integer value of 1 or -1, representing whether the house is to be bought or sold respectively

It returns:

- **Boolean:** True if the difference between the min and maximum number of houses in the colour group is less than 2. Else returns False.

*TurnFunctions.**firebase_house(player, player_ind, buysell, propIndex, prop, houseOrHotel, n)***: Prepares necessary strings for firebase implementation of buying and selling houses.

Takes in

- **player**: Instance of the ClassInit.Player class
- **player_ind**: Index of player within player_list, which is in turn obtained by GameInit.player_init()
- **buysell**: String of 'B' or 'S', representing the intention to buy or sell houses respectively.
- **propIndex**: Index of the specified instance of the ClassInit.Property class, within player's properties attribute list.
- **prop**: Instance of the ClassInit.Property class corresponding to the above propIndex.
- **houseOrHotel**: String of 'house' or 'hotel', representing whether to buy a house or a hotel respectively. Defaults to 'house'.
- **n**: String representing an integer value of the number of houses to be purchased. Defaults to a blank string.

It returns:

- **string:** String to be uploaded to firebase to be printed by in the consoles of players not taking the turn.
- **cmd:** String to be uploaded to firebase to be evaluated by in the consoles of players not taking the turn.

*TurnFunctions.**Win_Lose(player_list, player_name, false_players, Bank)***: Iterates through every player and check the victory condition. Ends the game once a winner is found.

Takes in

- **player_list**: List of all ClassInit.Player instances in order of turns.
- **player_name**: Name of the current player.
- **false_players**: List of all ClassInit.Player instances who lost.
- **Bank**: ClassInit.Player instance representing the Bank, created from GameInit.board_init().

It returns:

- **false_players:** An updated list of all ClassInit.Player instances who lost.

Once a winner is determined, the function prints the necessary graphics and ends the game using exit().

*TurnFunctions.**check_go(initial_pos, final_pos, player, board)***: Checks whether the player passed GO, and performs the necessary actions if so.

Takes in

- **initial_pos**: Integer representing initial player index
- **final_pos**: Integer representing initial player index
- **player**: Instance of the ClassInit.Player class
- **board**: List of property objects

It returns None, and if player is found to have passed index 0 of board, adds 200 to player's money attribute, and prints graphics, including Graphics.go.

*TurnFunctions.**move_board(player, dice_roll, board, move_str)***: Performs the movement of the player across the board.

Takes in

- **player:** Instance of the ClassInit.Player class.
- **dice_roll**: Integer, sum of both dice rolls.
- **board**: List of property objects
- **move_str**: String to be printed, which tells how many steps the player has moved.

It returns, in order, as a tuple:

- **pos:** The ClassInit.Property instance of the player's final position.

This function runs Graphics.Board_Animation() as well as TurnFunctions.check_go(), printing and updating states that these functions concern.

*TurnFunctions.**chance_card(card, player)***: Enacts the effect of the drawn chance card.

Takes in

- **card**: Tuple containing a string to be printed at index 0, and a integer value at index 1.
- **player:** Instance of the ClassInit.Player class.

It returns nothing, but prints card[0] and implements adds card[1] to player's money attribute.

*TurnFunctions.**cheatcode(player_ind, cheat, player_list, board)***: Performs the movement of the player across the board.

Takes in

- **player_ind:** Index of player within player_list, which is in turn obtained by GameInit.player_init()
- **cheat**: String representing cheat to implement, or None.
- **player_list**: List of all ClassInit.Player instances in order of turns.
- **board**: List of property objects.

It returns:

- **cheat:** If cheat == None, cheat is asked for as an input, and will be returned as a string. The purpose of this is for the firebase implementation.

Effect implemented will depend on the cheat string passed:
If cheat == 'WIN', make all other players except the current player bankrupt.
If cheat == 'LOSE', make the current player bankrupt.
If cheat == 'GODMODE', make the current player the owner of all properties, adds 5 houses to each property, and adds 10000000 to the current player's money attribute.

Main.py

Dependencies: os library, time library, random library, pyrebase library, TurnFunction.py, GameInit.py, Graphics.py

Functions

*Main.**upload_ firebase(use_firebase, string, cmd)***: Uploads necessary data to firebase.
Takes in

- **use_firebase:** Boolean value to determine online (True) or offline (False) play.
- **string:** String to be uploaded to firebase to be printed by in the consoles of players not taking the turn. Defaults to None.
- **cmd:** String to be uploaded to firebase to be evaluated by in the consoles of players not taking the turn. Defaults to None.

It returns nothing, and uploads string to cmd according to the firebase keys of the same name, if use_firebase is True.

*Main.**PrintBoard(board, player)***: Prints the board in a readable format
Takes in

- **board:** List of property objects.
- **player:** Instance of the ClassInit.Player class.

It returns nothing, and prints the player's money attribute as well as the board in a readable format.

*Main.**turn(player, board, player_ind, use_firebase, Bank, player_names)***: Runs the turn of the current player.
Takes in

- **player:** Instance of the ClassInit.Player class.
- **board:** List of property objects.
- **player_ind:** Index of player within player_list, which is in turn obtained by GameInit.player_init()
- **use_firebase:** Boolean value to determine online (True) or offline (False) play.
- **Bank:** ClassInit.Player instance representing the Bank, created from GameInit.board_init().
- **player_names:** List of all player name attributes, in turn order.

It returns nothing, instead hosting each phase of a turn for each player. Each phase is explained in greater detail in the code description. The pseudo-code is as follows:

1. Initialize turn
2. Asks for input for phase 1 - Roll Dice, View Board, View your Properties.

    a. While action is not to roll the dice, repeat 2.
    b. If action is to roll dice, run TurnFunctions.move_board.
3. Board tile interactions
    a. Change state of the player and board depending on what tile the player lands on.
    b. Interacts with the tiles according to player input if needed.
4. Other turn actions - Build/Sell Houses, Trade, Mortgage, View Board, End turn.
    a. Asks for input for above actions
        i. Implements actions according to input.
    b. While action is not to end turn, repeat 4.

Throughout the function, if use_firebase is True, the function will update firebase with the necessary data for the significant actions performed.

*Main.**main()***: Runs the main loop of the game.

Does not take in arguments or return. The pseudo-code is as follows:

1. Initializes the main menu. Asks for mode of play
2. Asks for input for player number and names for player initialization.
3. Once above variables are set, GameInit functions are called and initialization completes.
4. Main game loop starts.
    a. Loop changes depending on the initialized use_firebase Boolean.
        i. If offline, each time a turn ends, the loop loops.
        ii. If online, each loop:
            1. Corresponds to a single player's turn, and will loop once after that player has completed a turn
            2. Contains another while loop that breaks once the turn variable on firebase becomes that player's turn. Within the loop, the function will constantly check the necessary firebase variables to print new strings and run new commands.
        iii. The function then checks for the defeat/victory of each player at the end of every player's turn.
    b. Loop does not break. Instead, program ends via exit() by TurnFunctions.Win_Lose().