Design Overview for Monopoly

Name: Nguyen Gia Binh
Student ID: 104219428

Summary of Program

This is a basic, extra simple of the game Monopoly. User will experience the game through a simple text console

Starting up the program, a board is created using a script with 16 squares. Then the program ask for the players (2-4 players) info. After that, first player and so on after the one in front finished will have a list of commands containing actions that they can perform like if they land on a property then do they want to buy it or skip and so on with differents kinds of square having different effect. Other than "property" square, there are "card" square containing effect like moveforward/backward, plus/minus money, go to prison for tax evasion to make the game more challenging. If no one forfeit then player that can't afford to pay rent lose and the game is "the last one standing" game.

Required Roles

Responsibility	Type Details
Program	Its oonly job is to start the game by calling the Initialize function and then Command line for player for the first time from GameLogic and GameLogic will handle for the rest of the program
GameLogic	This is where the program calculate money, printing feedback, executing command, checking if anyone lose or win, processing square acquisition and upgrade, checking for position and state of the game.
Square	The position of squares which other class will use
Property	This class enable the program to manipulate the property on some element but mostly it is for reading the property info which will be pre-manufactured
Board	This is where all the info of each square is stored and communicate, allowing for other to get the information of the square and manipulate it as fit
PropertyState	Main purpose of this class is so that property can be marked Bought, Rent,…. If a player land on it and got to do something with the square. It is also an abstract class to allow other "State" class to implement there own way of calculating fee and such
Card	This class manage the card square, in the original board game there are 2 type of card Chance and Community_Chest, each have similar and different instructions. For simplification the only thing separate this 2 type of card is the name and type, the effect stay the same
Player	Managing player info, player can have 3 type of state: win/lose, in prison or not and have prison pass or not much like accountant this day. This class also contain function associate with the player like rolling dice cuz the original game make the

	players roll dice themselves, the function for moving forward/backward is also in player class
BoughtStatus	Inherit PropertyState to be able to Set the property to Bought status which correspond to level 1 square and have its own price calculation
LandStatus	Inherit BoughtStatus to be able to Set the property to Land status which correspond to level 2 square and have its own price calculation. Make this inherit from bought status instead of property status make it so that the only way for it to be able to get to lv2 is to get to lv1 first
RentStatus	Inherit LandStatus to be able to Set the property to Rent status which correspond to level 3 square and have its own price calculation. Make this inherit from land status instead of property status make it so that the only way for it to be able to get to lv3 is to get to lv2 first
SquareFetch	An abstract class use to locate the card suqare position when a player land on it. Can be use for more like Splashkit but due to circumstances splashkit is no longer a part of this project
CardFetch	A constructor to set the type of the card and know the card position by inheriting from SquareFetch

Game Function:

- Initialize(): start the game by asking for player name and put it in a list

- Game_main(): execute every turn, announce who turn it is and roll the dice, check for winner and loser

- Win(): if player money is not 0 and player has not FF then count player as winner, many winner until 1 remain. To double check for number of winner we create a lose condition where it count number of winner until 1

- Game_choice(): A command list containing everything a player is allow to do including thing that they are not able to do yet, there will be messages indicating player cant do certain action yet until condition met

- PlayerProfile(): print out the current player information and position

- Currentposition(): Check if player is in a square that have been bought/upgrade by other player and if it is then player will have to pay a certain amount(not enough money = lose). This function also print out the square information. It also check if this square is a card square or government own square

- EmptyPosition(): empty position are square that are left blank for starting point, visiting prison and go to prison without getting the benefit of +300 when pass the starting point

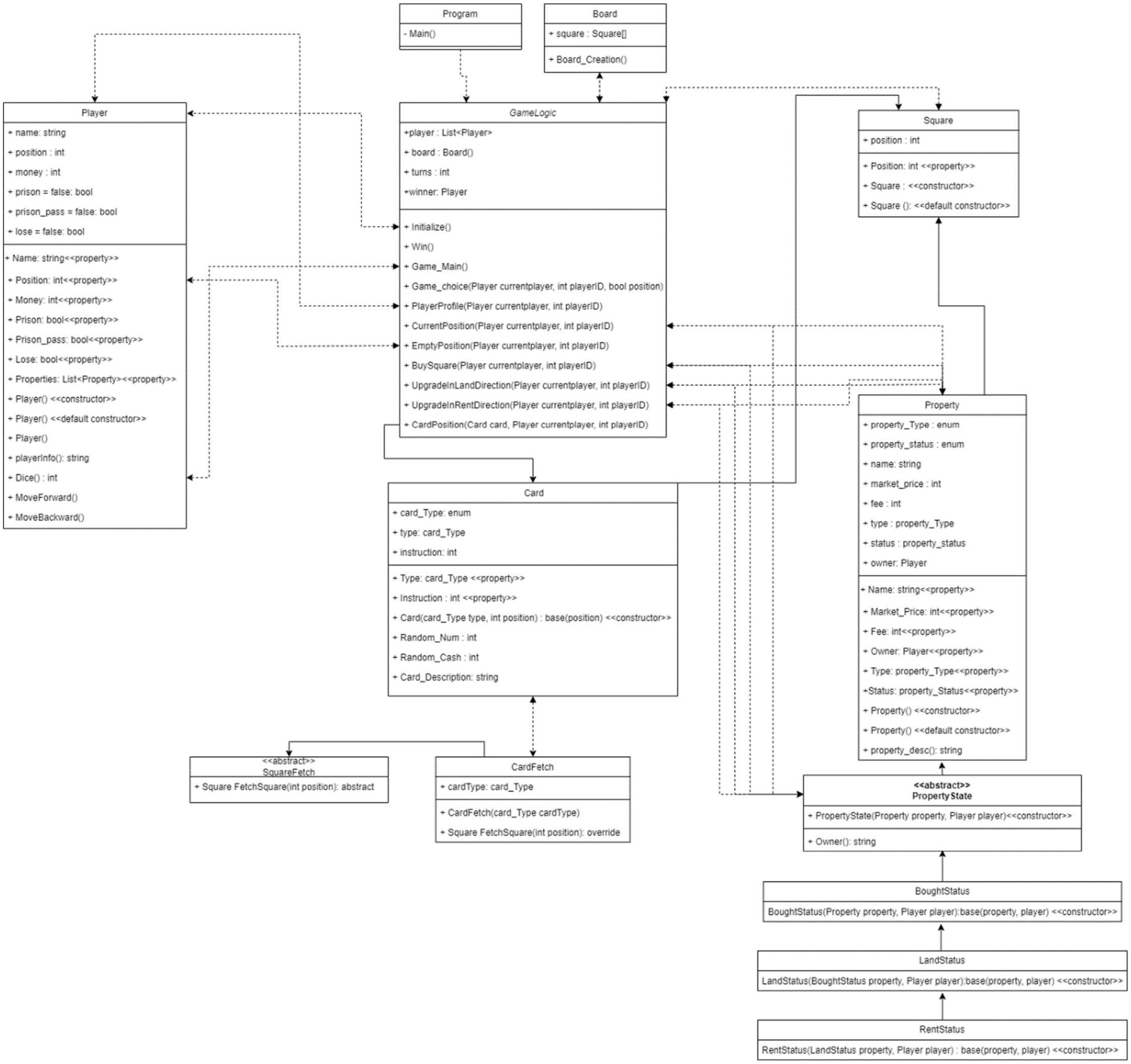
- BuySquare(): if the square has no owner then player can buy it. A message will pop up if player doesn’t have enough money to make the purchase. If the square is blank then it is government property

- UpgradeInLandDirection(): Count the number of bought property to upgrade to lv 2. If already lv 2 then it stop and prompt the user to go back to Game_choice or choose another property. It can also check if player has enough money to go through with the upgrade

- UpgradeInRentDirection(): Count the number of bought property to upgrade to lv 3. If already lv 3 or only at lv 1 then it stop and prompt the user to go back to Game_choice or choose another property. It can also check if player has enough money to go through with the upgrade

-cardPosition(): Base on which card the player get (random btw) then the corresponding effect is applied

Class Diagram


Program
- Main()
Board
+ square : Square[]
+ Board_Creation()
Player
+ name: string
+ position : int
+ money : int
+ prison = false: bool
+ prison_pass = false: bool
+ lose = false: bool
+ Name: string<<property>>
+ Position: int<<property>>
+ Money: int<<property>>
+ Prison: bool<<property>>
+ Prison_pass: bool<<property>>
+ Lose: bool<<property>>
+ Properties: List<Property><<property>>
+ Player() <<constructor>>
+ Player() <<default constructor>>
+ Player()
+ playerInfo(): string
+ Dice() : int
+ MoveForward()
+ MoveBackward()
GameLogic
+player : List<Player>
+ board : Board()
+ turns : int
+winner: Player
+ Initialize()
+ Win()
+ Game_Main()
+ Game_choice(Player currentplayer, int playerID, bool position)
+ PlayerProfile(Player currentplayer, int playerID)
+ CurrentPosition(Player currentplayer, int playerID)
+ EmptyPosition(Player currentplayer, int playerID)
+ BuySquare(Player currentplayer, int playerID)
+ UpgradeInLandDirection(Player currentplayer, int playerID)
+ UpgradeInRentDirection(Player currentplayer, int playerID)
+ CardPosition(Card card, Player currentplayer, int playerID)
Square
+ position : int
+ Position: int <<property>>
+ Square : <<constructor>>
+ Square (): <<default constructor>>
Card
+ card_Type: enum
+ type: card_Type
+ instruction: int
+ Type: card_Type <<property>>
+ Instruction : int <<property>>
+ Card(card_Type type, int position) : base(position) <<constructor>>
+ Random_Num : int
+ Random_Cash : int
+ Card_Description: string
Property
+ property_Type : enum
+ property_status : enum
+ name: string
+ market_price : int
+ fee : int
+ type : property_Type
+ status : property_status
+ owner: Player
+ Name: string<<property>>
+ Market_Price: int<<property>>
+ Fee: int<<property>>
+ Owner: Player<<property>>
+ Type: property_Type<<property>>
+Status: property_Status<<property>>
+ Property() <<constructor>>
+ Property() <<default constructor>>
+ property_desc(): string
<<abstract>>
SquareFetch
+ Square FetchSquare(int position): abstract
CardFetch
+ cardType: card_Type
+ CardFetch(card_Type cardType)
+ Square FetchSquare(int position): override
<<abstract>>
Property State
+ PropertyState(Property property, Player player)<<constructor>>
+ Owner(): string
BoughtStatus
BoughtStatus(Property property, Player player):base(property, player) <<constructor>>
LandStatus
LandStatus(BoughtStatus property, Player player):base(property, player) <<constructor>>
RentStatus
RentStatus(LandStatus property, Player player) : base(property, player) <<constructor>>