Aditya Kotak, Rahul Sarathy, William Wang

Mr. Peck

APCS, Period 6

8 May 2015

Preliminary Specification for Monopoly Final Project

**I: Structural Design**

*GameBoard Class*

|  |  |
| --- | --- |
| **Data** |  |
| GameBoard | Circular LinkedList (ListNode2) => ListNode2<Space> |
| Chance cards | Queue => LinkedList<Chance> |
| Community Chest cards | Queue => LinkedList<CommunityChest> |
| List of players | Queue => LinkedList<Player> |

*Player Class*

|  |  |
| --- | --- |
| **Data** |  |
| List of properties | List => ArrayList<PropertySpace> |

*Space Class (Abstract)*

|  |  |
| --- | --- |
| **Data** |  |
| List of players | Queue => LinkedList<Players> |

**II: Object-Oriented Design**

MonopolyGame is the main class that will run the entire program. It contains Card objects, Player objects, a Bank object, a GameBoard object, two Die objects, and Building objects.

A Card object represents a card in the game, of which there are two types, Chance and CommunityChest. It is able to be dealt, and has a general function to allow it to execute actions based upon what each card is supposed to do.

A Player object represents one player of the game. It has variables to store the player’s name, total cash, total number of houses, total number of hotels, the total worth (cash + property + buildings), list of properties owned, as well as whether or not the player is in jail, whether or not the player has a “get-out-of-jail-free card”, whether or not the player has money to make a purchase, whether or not the player has any cash left, and where the player is on the board. A player can roll dice, move, get information about the space it is on, draw either a Chance or a CommunityChest, buy and sell buildings and properties, pay and receive money from the bank and other players, and mortgage its properties if it has them and needs cash.

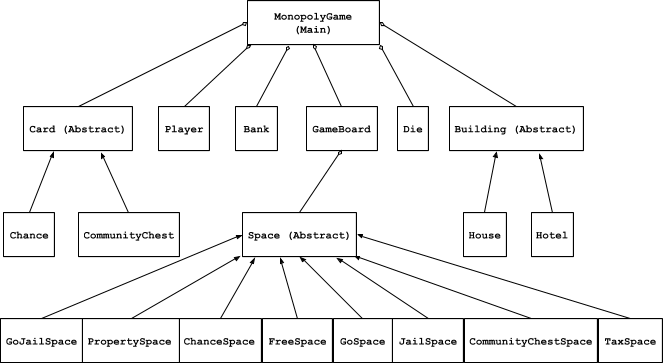
A Bank object manages the cash transactions of the game. Before the game, it gives each player a preset amount of cash to start with. During the game, the bank is able to pay players and get paid by players.

A Die object models a 6-sided die in the game. It generates a number from 1 to 6 inclusive, which helps determine events like player movement, how many times a player can roll, whether or not the player goes to jail, and whether or not the player is freed from jail.

A Building object represents a generic building in the game, of which there are two types, House and Hotel. Both types can be bought or sold, but you must have bought 4 houses before you are allowed to buy 1 hotel.

A GameBoard object represents the gameboard on which the players will be moving around and interacting with the Space objects that make it up. It is represented by a circular linked list of Space objects , allowing continuous circulation around the board.

A Space object represents one generic space of a GameBoard. There are 8 different types of Space object, each of which has a slightly different function, but all generic spaces have a name, a location on the board, whether or not it can be bought, whether or not it can have buildings on it, and a list of players occupying it.



**III: Detailed Design**

While we have not done documentation yet for the classes, we have listed out all the methods and variables needed in our project. Here they are:

|  |
| --- |
| MonopolyGame (Main)   * public static void main(String[] args)   \*Player Class   * String name; * int cash, netWorth; * boolean hasJailFree, inJail; * boolean isBroke; * boolean enoughMoney; * ArrayList<PropertySpace> properties; * int numHouses, numHotels; * int position; * public int rollDie() * public void move(int numOfSpaces) * public void getSpaceInfo(Space s) * public void drawChance() * public void drawCC() * public void buy(Building b) * public void buy(Space s) * public void sell(Building b) * public void sell(Space s) * public void mortgage() * public void receiveFromBank(int money) * public void receiveFromPlayer(int money) * public void payPlayer(int money) * public void payBank(int money)   \*Space Abstract Class   * String name; * int loc; * final boolean buyable; * final boolean canHaveBuildings; * Queue<Player> playersOnSpace;   \*GameBoard Class   * Queue<Player> players; * Die d1, d2; * Queue<Chance> ch; * Queue<CommunityChest> cc; * LinkedList<Space> board; * int d1Val, d2Val; * int numOfDoubles; * public void setup() * public void getSpaceInfo(Player p) * public int move(Player p, int numOfSpaces) * public Chance drawChance() * public CommunityChest drawCC() * public int rollDie()   Building Abstract Class   * public int getPrice() * public int getRent()   Bank Class   * public payPlayer(int amount, Player name) * public receiveMoney (int amount)   Chance Class (super)   * private int cardID; // 0-16 * public void drawChance(int id)   + switch(id) * public int getCardID() * public void performAction()   CommunityChest Class (super)   * private int cardID; // 0-16 * public void drawChance(int id)   + switch(id) * public int getCardID() * public void performAction()   Die (class)   * public void roll(); * public int getRoll1(); * int int getRoll2();   House (class, extends Building)   * int numOfHouses; * boolean allProperties; * public void buyHouse();   Hotel (class, extends Building)   * boolean allProperties, maxHouses; * public void buyHouse();   JailSpace (class, extends Space)   * int tries; * boolean paid, inJail, rolledDouble;   GoJailSpace (class, extends Space)   * public void moveToJail(Player one)   GoSpace (class, extends Space)   * public void moveToSpace(Player one, Space loc)   \*PropertySpace (class, extends Space)   * int houseCount; * ArrayList<House> houses; * Hotel hotel; * int rent; * int spacePrice; * int housePrice; * int hotelPrice; * boolean hasHouses; * boolean hasHotel; * Player owner; * public Player getOwner()   FreeSpace (class, extends Space)  TaxSpace (class, extends Space)   * public int getTax() |

**IV: Testing**

 Possible issues and bugs that can occur:

A JUnitTest will have to test for several different scenarios in which we anticipate the most problems. We have provided strategies to test for some features which we believe to be most likely to contain bugs.

* Does a player receive money after passing GO?
  + Check a player’s balance before and after go and confirm that 200 has been added after passing GO
* Transferring assets after a player goes bankrupt
  + Confirm that if player goes bankrupt, assets are sold to (and owned by) bank until player has at least 200
* Converting from a regular property to a property with houses to one with a hotel
  + Confirm that while a hotel exists on a property, there are no houses on that property
  + Confirm that the proper rent corresponding to the number of houses is being charged
* Staying in jail for the correct amount of time
  + Allowing the user to leave if they pay bail, or if they roll doubles
  + Confirm that the player stays in jail for a maximum of 3 turns
* Correctly implementing a double roll (if doubles allowing for an extra turn)
  + Confirm that if person A rolls two of the same number, person A is granted an extra turn
  + Confirm that if person A rolls doubles 3 times in a row, they will be sent to jail