



Presented to the Software Technology Department  
De La Salle University - Manila  
Term 3, A.Y. 2018-2019

In partial fulfillment

of the course

**In Object-Oriented Design and Programming (S16)**

**MYEMPIRE MACHINE PROJECT TEST CASES**

Submitted by:

Pelagio, Trisha  
Ramirez, Bryce Anthony

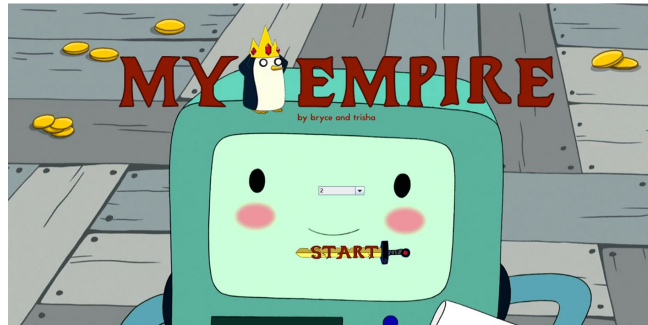
Submitted to:

Ms. Shirley Chu

## Starting Screen

Expected: The program will ask for the number of players.

Actual:



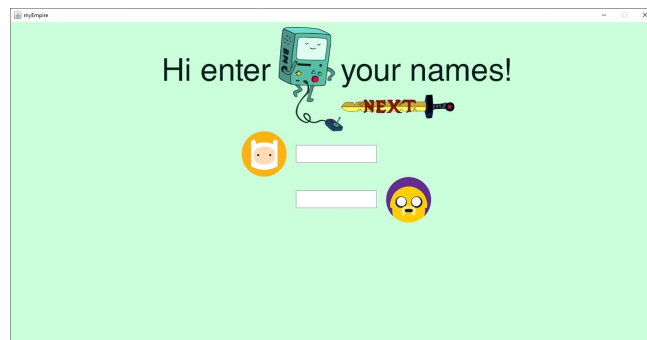
## Board

### 1. Public Board (int totalPlayer)

#### a. If total player selected is equal to 2:

Expected : The program will start with 2 players and players will be asked for names.

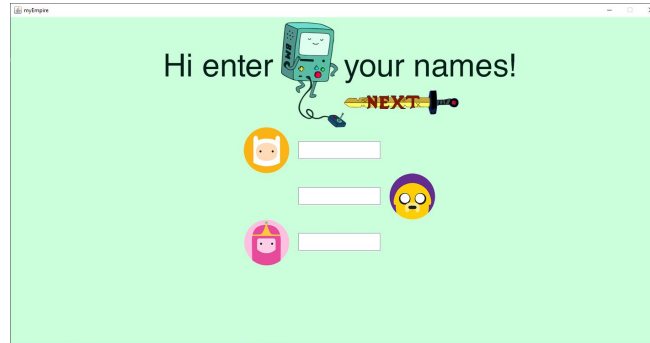
Actual : The game started with two players and asked for their names.



#### b. If total player selected is equal to 3:

Expected : The program will start with 3 players and players will be asked for names.

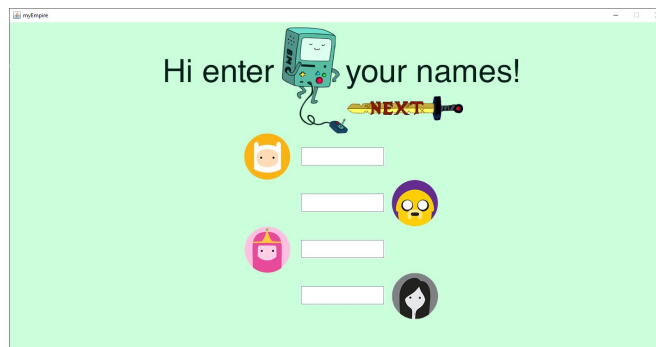
Actual : The game started with 3 players and asked for their names.



- c. If total player selected is equal to 4:

Expected : The program will start with 4 players and players will be asked for names.

Actual : The program started with 4 players and asked for their names.

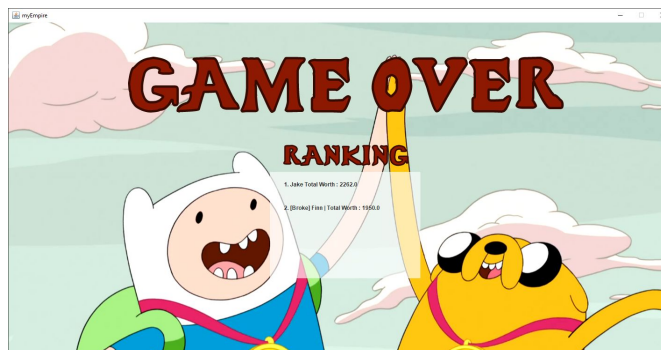


## 2. Public boolean isEndGame (Bank bank)

- a. If at least one of the Players is broke which is identified through playerBroke() :

Expected: The function immediately returns true. The game will end.

Actual : The game ended.



- b. If a Player has reached 2 full sets of house. This value is determined through the return value of getFullSet(). :

Expected: The function immediately returns true. The game will end.

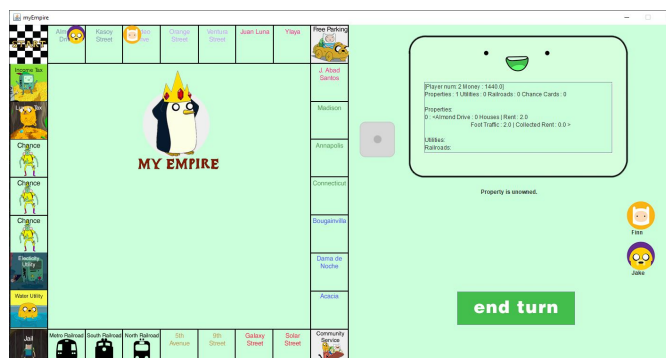
Actual : The game ended.



- c. If the Bank, specified in the parameter, runs out of money. :  
Expected: The function immediately returns true. The game will end.  
Actual: The game ended.



- d. If none of the winning conditions are met:  
Expected: The function returns false. Game will continue.  
Actual: The game did not end.



### 3. Public Player[] getWinners ()

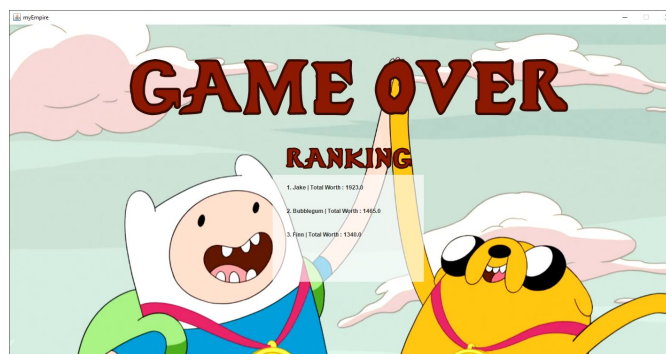
- a. If a player runs out of balance :  
Expected: The function will return an array of players with bankrupt players at the bottom, ranked by their worth and the non-bankrupt players ranked at the top.  
Actual: The game ended with bankrupt player at the bottom.



- b. If Bank runs out of balance :

Expected: The function will return an array of players arranged in descending order, index 0 will contain the first place and last index for the last place.

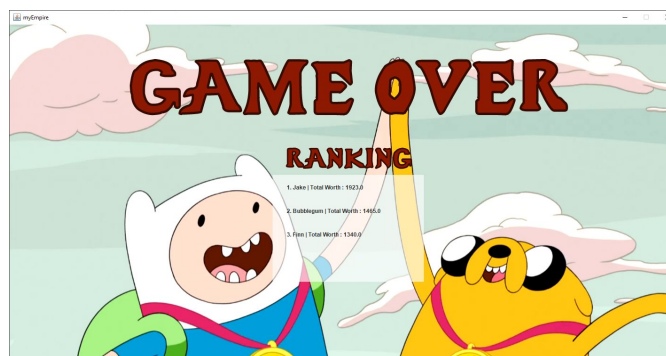
Actual: The game ended ranking players in descending order based on total worth.



- c. If a player has reached 2 full sets of house :

Expected: The function will return an array of players arranged in descending order, index 0 will contain the first place and last index for the last place.

Actual: The game ended ranking players in descending order based on total worth.



4. Public double playerTotalWorth(Player player)

- a. If player does not have property, railroad, and utility :  
Expected: The function only returns the balance of a Player specified in the parameter.  
Actual: Player's total worth is only equal to its balance.



- b. If player only has railroads:  
Expected: The function only returns the balance of a Player added with the worth of all owned railroads  
Actual: Player's total worth is equal to balance plus worth of owned railroad.



- c. If player only has properties:  
Expected: The function only returns the balance of a Player added with the worth of all owned properties.  
Actual: Player's total worth is equal to balance plus worth of owned properties.



- d. If player only has utilities:

Expected: The function only returns the balance of a Player added with the worth of all owned utilities.

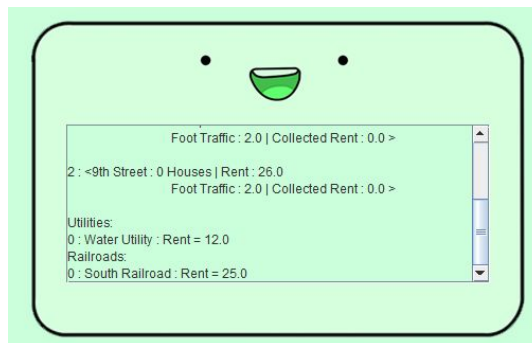
Actual: Player's total worth is equal to balance plus worth of owned utilities.



- e. If player contains properties, railroads, and utilities:

Expected: playerTotalWorth() obtains the sum of the worth of all properties, railroads, and utilities owned by the Player, and also includes the balance property.

Actual: Player's total worth is equal to balance plus worth of owned railroads, utilities and properties.

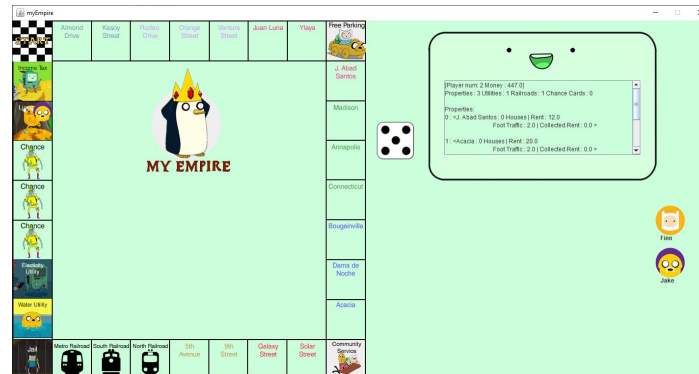


## 5. Public void nextTurn()

- a. If turnIndex is not at the last possible value :

Expected: The turnIndex is incremented by 1 in the function.

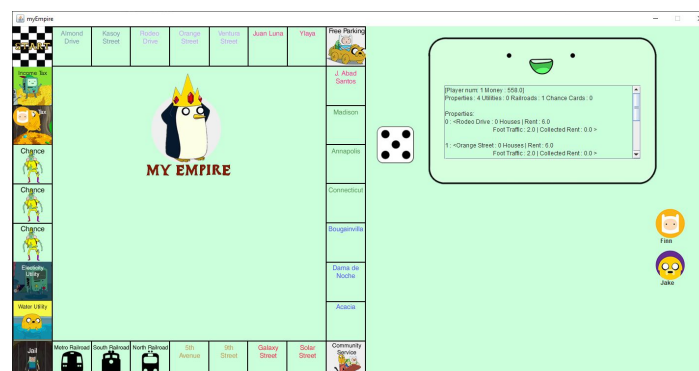
Actual: The turn is incremented by 1.



b. If turn index is equal to totalPlayer :

Expected: The value of turnIndex resets to 0 after incrementing by 1 due to the modulo operator

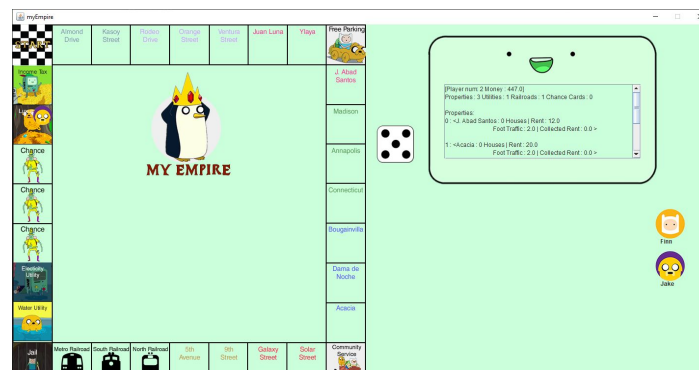
Actual: The turn reset to 0. Back to Player 1's turn.



c. If turn index is at the starting index:

Expected: The turnIndex is incremented by 1 in the function.

Actual: Turn is incremented by 1.



## Player

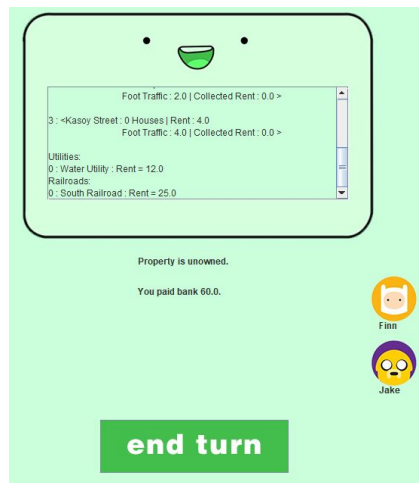
1. public boolean purchaseSpace (Space space, Bank b)

a. If Space type is Property and can be bought because valid transaction :

Expected: Player will own the property and it will be added to array list of properties, player will pay bank and property price will be updated to



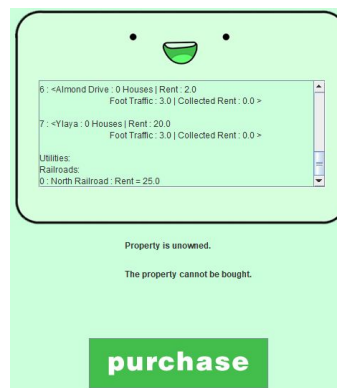
check for similar property types and corresponding prices per number of property owned of the same color.  
Actual: The property was bought.



- b. If Space type is Property and cannot be bought because insufficient money :

Expected: Player will not own the property. There will be a screen output saying that property cannot be bought.

Actual: The property cannot be bought.



- c. If Space type is Utility and can be bought because valid transaction:

Expected: Player will own the utility and it will be added to array list of utilities, player will pay the bank and utility price will be updated depending on the number of other utilities owned times the number shown on the dice.

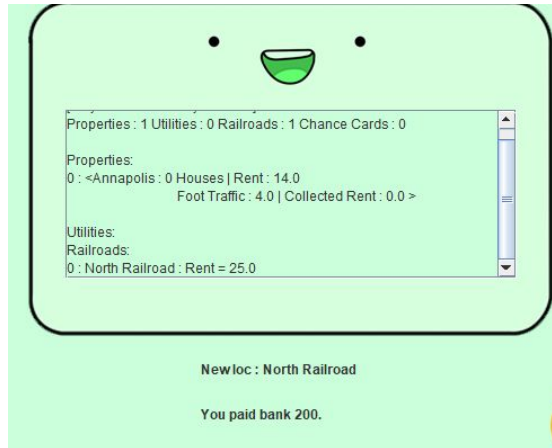
Actual: Utility bought. It is updated by 1 (number shown on dice) times 4.



- d. If Space type is Utility and cannot be bought because insufficient money:  
 Expected: Player will not own utility and there will be a screen output saying that the utility cannot be bought.  
 Actual: The utility cannot be bought.



- e. If Space type is Railroad and can be bought because valid transaction:  
 Expected: Player will own the railroad and it will be added to array list of railroads, player will pay the bank and railroad price will be updated depending on the number of other railroads owned.  
 Actual: The rent of the railroad is updated to 25 because player owns only one.



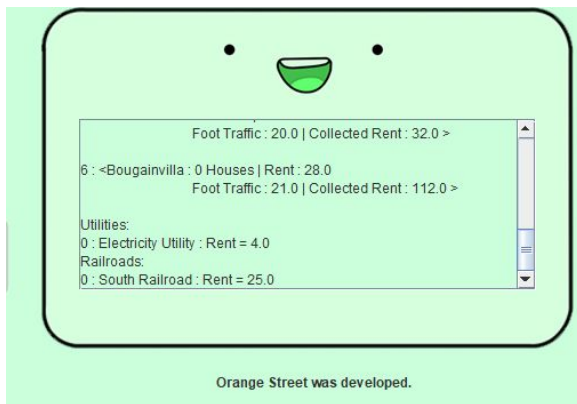
- f. If Space type is Railroad and cannot be bought because insufficient money:  
Expected: Player will not own railroad and there will be a screen output saying that the railroad cannot be bought.  
Actual:
2. public boolean developProperty(Space property, Bank b)  
a. If Property cannot be developed due to insufficient balance :  
Expected: There will be a screen output saying that the property cannot be developed.  
Actual: The property was not developed



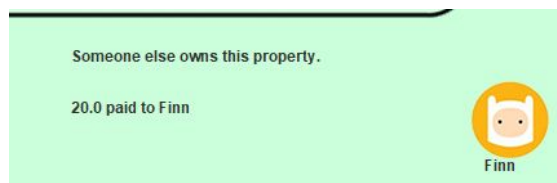
- b. If Property cannot be developed because Property does not have enough Foot Traffic and Property does not have enough collected rent :  
Expected: There will be a screen output saying that the property cannot be developed.  
Actual: The property was not developed



- c. If Property has enough money and Property can be developed:  
 Expected: Property will be developed and player will pay the bank with the amount needed for development and program will also check for fullset properties after development.  
 Actual: The property was developed



3. public void payPerson(double amount, Player person)  
 a. If Player paid a person with money balance greater than amount :  
 Expected: Player's money will get subtracted with that amount.  
 Actual:



- b. If Player paid a person with insufficient money balance :  
 Expected: Player's money will become 0 because player cannot pay anymore (Bankrupt) and the game will end.  
 Actual: The game ended



- c. If Player paid a person with money equal to the amount:  
Expected: Player's money will get subtracted with that amount. Game will not end until player cannot pay anymore.

Actual:

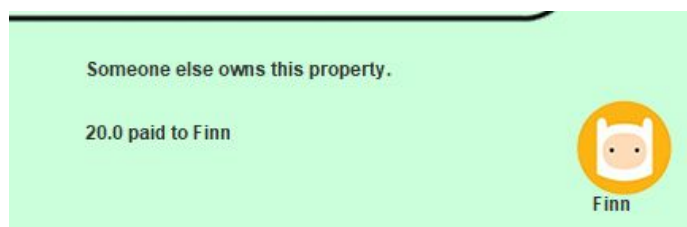


#### 4. public void receiveMoney (double amount)

- a. If Player received money from Person who has greater money balance than amount to be paid :

Expected: Player will receive that amount of money, the game continues.

Actual:



- b. If Player received money from Person who does not have enough money balance to pay :

Expected: The game will end.

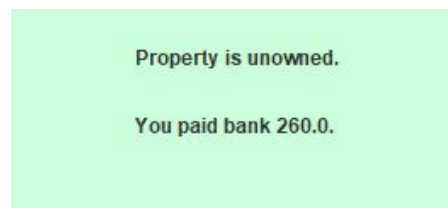
Actual:



- c. If Player received money from Person who has equal money balance to amount to be paid :  
 Expected: Player will receive that amount of money, the game continues.  
 Actual: Game continues



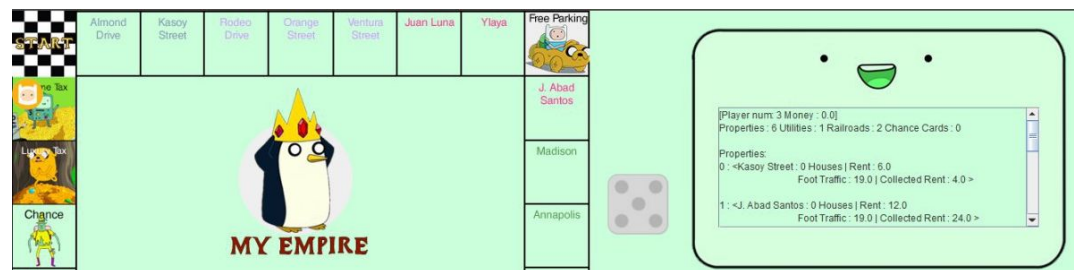
5. public void payBank (double amount, Bank b)  
 a. If Player paid a bank with money balance greater than amount :  
 Expected: Bank will receive that amount and the game continues.  
 Actual:



- b. If Player paid a bank with insufficient money balance:  
 Expected: The game will end.  
 Actual: The game ended.



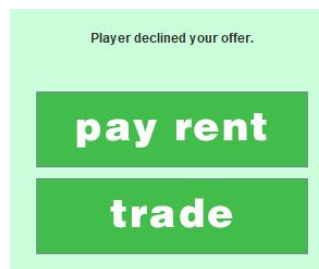
- c. If Player paid a bank with money equal to the amount:  
 Expected: Bank receives that amount and the game continues.  
 Actual: The game continued



6. public void trade(Player with, Space property)  
 a. If other player accepts trade:  
 Expected: The properties chosen will exchange owners.  
 Actual: The properties exchanged owners. Trade was successful.



- b. If the other player does not accept trade:  
 Expected: The trade will be unsuccessful  
 Actual: The trade was unsuccessful.



- c. If user inputted invalid index:  
 Expected: The player will be asked again for an input  
 Actual : The player had to enter again an index until input is valid.



7. private boolean isFullSet(Space property)

- a. If Player does not have properties:

Expected: The fullset variable will not be incremented.

Actual: Fullset variable is not incremented. (No GUI Implementation)

- b. If Player has properties but is not full set:

Expected: The fullset variable will not be incremented.

Actual: Fullset variable was not incremented. (No GUI Implementation)

- c. If Player has properties and is full set:

Expected: The fullset variable will be incremented.

Actual: The fullset variable was incremented. (No GUI Implementation)

8. private void updateRent (Space space)

- a. If space type is Property and player has more than one of the same color:

Expected: The additional rent value will be changed to 10 times the additional properties owned by the player.

Actual: The additional rent value was changed to 10 times the additional properties owned by the player.

- b. If space type is Property and player has only one property of that color:

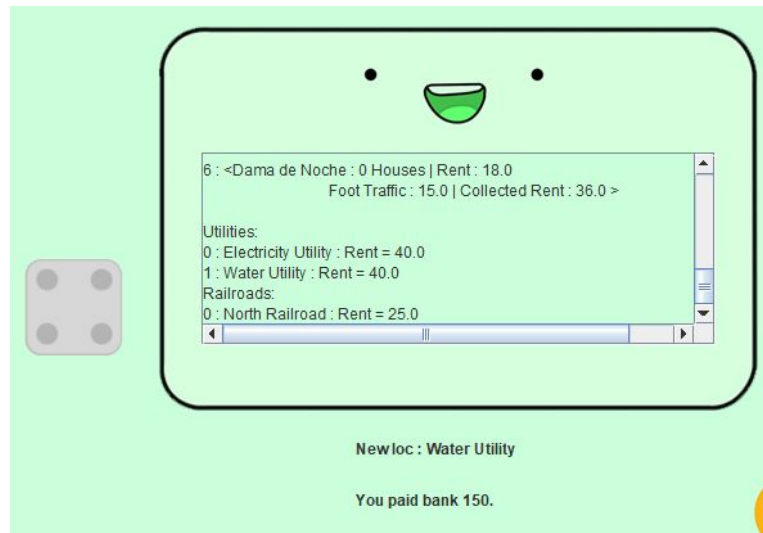
Expected: There will be no change in additional rent value.

Actual: There was no change in additional rent value.





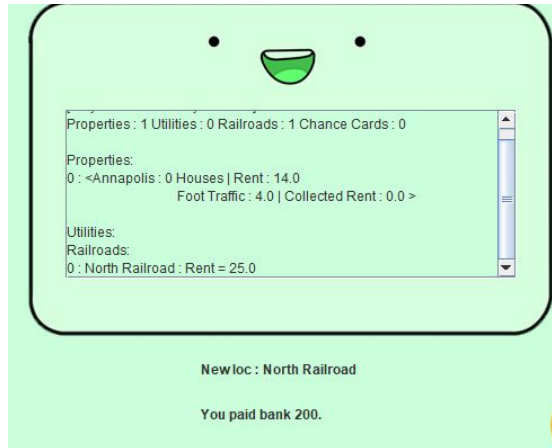
- c. If space type is Utility and player owns more than one utility:  
 Expected: Rent value will be updated to number shown in dice time 10.  
 Actual: Rent value was updated to number shown in dice time 10.



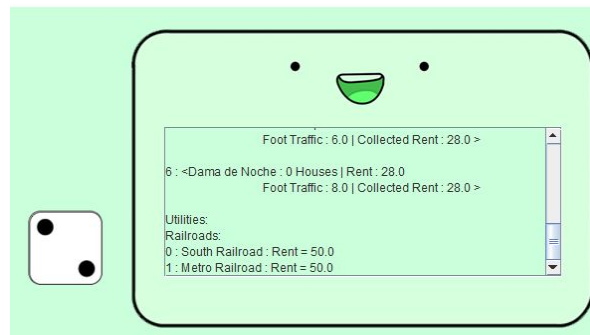
- d. If space type is Utility and player owns only one utility:  
 Expected: Rent value will be updated to number shown in dice times 4.  
 Actual: Rent value was updated to number shown in dice times 4.



- e. If space type is Railroad and player owns one railroad:  
 Expected: Rent value will be updated to 25.  
 Actual: Rent value was updated to 25.



- f. If space type is Railroad and player owns more than one railroad:  
Expected: Rent value will be updated to 50 or 150.  
Actual: Player owns 2 railroads, rent was updated to 50.

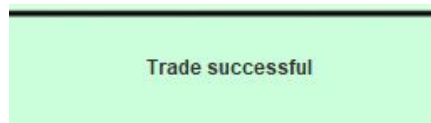


## 9. public boolean acceptTrade()

- a. If player accepted Trade (1) :

Expected: The properties will exchange owners and there will be a screen output indicating that the trade was successful.

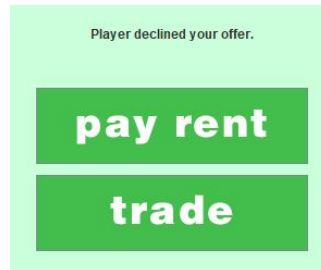
Actual: The properties exchanged owners and a screen output stated that trade was successful.



- b. If player did not accept trade (2) :

Expected: The properties will remain to their original owners and there will be a screen output indicating that the trade was not successful.

Actual: Trade was not successful.

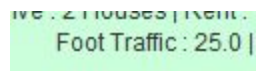


- c. If player entered wrong value :
- Expected: The user will be asked again for a value until valid input is entered.
- Actual:

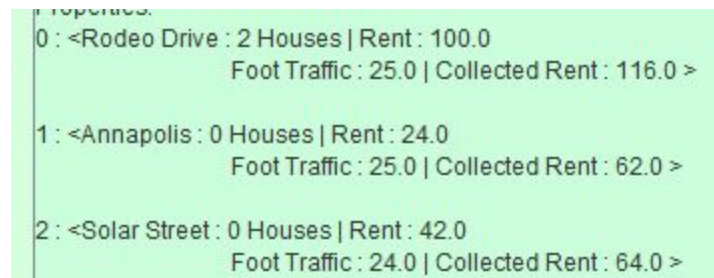


#### 10. public void addTraffics(Board board, int oldpos, int newpos)

- a. If player passed start:
- Expected: All property spaces passed from one space after the old position until new position will be incremented (Foot Traffic Count) including those after start.
- Actual:Foot traffics were incremented



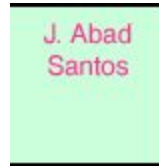
- b. If player did not pass start:
- Expected: All property spaces passed from one space after the old position until new position (Foot Traffic Count) will be incremented.
- Actual: All property spaces passed from one space after the old position until new position (Foot Traffic Count) were incremented.



- c. If no space type equal to property in between oldpos and newpos (inclusive):

Expected: No foot traffic will be incremented.

Actual: Foot traffic was not incremented



11. public boolean payerBroke()

- a. If money is equal to zero:

Expected: Player is not yet broke (was able to pay but now, money is 0).  
Game continues.

Actual: Game continued.



- b. If money is less than zero:

Actual: Player broke status is set to true and money is set to 0. Game ends.

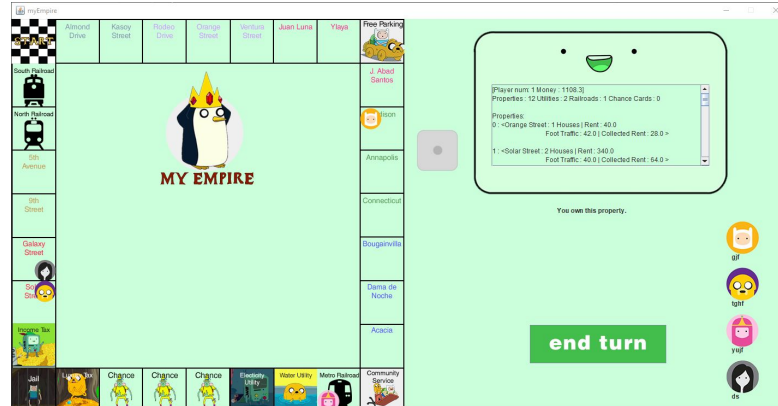
Expected: Game ended.



- c. If money is greater than zero:

Expected: Player is not broke, game continues.

Actual: Game continues



## Bank

1. Public void receiveMoney(double amount)
  - a. If Bank received money from Player who has greater money balance than amount to be paid :
 

Expected: The transaction becomes successful, and the amount from the parameter is added to the balance of the Bank.

Actual: Money was added to bank.
  - b. If Bank received money from Player who does not have enough money balance to pay:
 

Expected: The game will end.

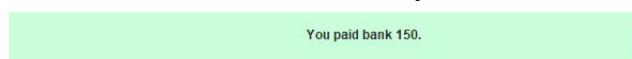
Actual: The game ended.



- c. If Bank received money from Player who has equal money balance to amount to be paid:
 

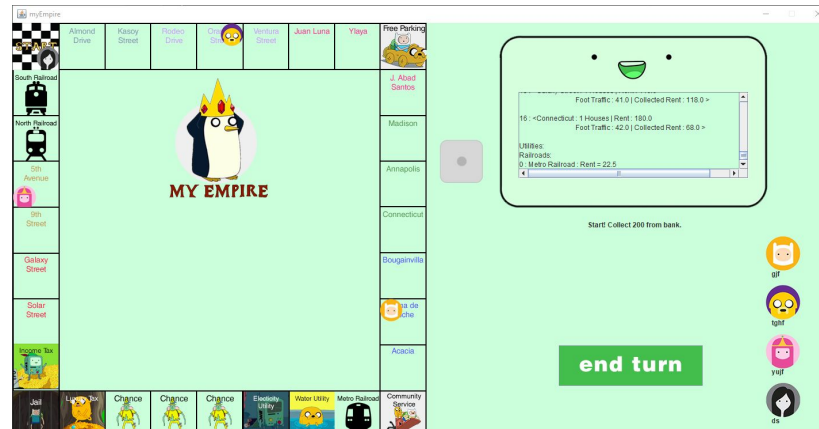
Expected: The transaction is still successful, therefore the amount is added to the balance of the Bank.

Actual: The transaction is successful, Money was still added to bank.

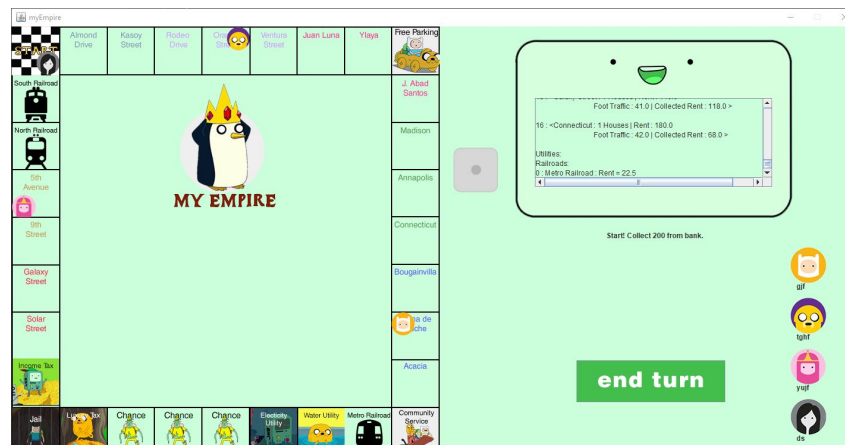


2. Public void subtractMoney(double amount)

- a. If Bank deducts money less than its balance :  
 Expected: The process is valid which results to subtracting the amount identified in the parameter to the balance of the Bank instance.  
 Actual: Transaction valid, game continues.



- b. If Bank deducts money equal to its balance  
 Expected: The condition is still valid and also subtracts the amount in the parameter from the balance of the Bank.  
 Actual: Transaction is valid, game continues.



- c. If Bank deducts money greater than its balance :  
 Expected: The process is not valid and the game will end.  
 Actual: Game ended.



3. Public void addTo(Player p, double amount)

- a. If Bank gives money less than its balance

Expected: The bank adds money to the Player object equal to the amount specified in its parameter.

Actual: The bank added money to player.

Start! Collect 200 from bank.

- b. If Bank gives money equal to its balance

Expected: The process is still valid, therefore performs the operations of subtracting the amount in the parameter from the balance and adding it to the balance of a specified Player.

Actual: Bank successfully added money to player. Game did not end.

Start! Collect 200 from bank.

- c. If Bank gives money greater than its balance

Expected: The game ends because bank is out of money.

Actual: Game ended.



## Money

1. Public boolean isValidTransaction(double amount)
  - a. If the balance of the object is greater than the value indicated in the parameter :  
Expected: The function returns a boolean value of true.  
Actual: The function returned true. (No GUI implementation)
  - b. If the balance of the object is equal to the value indicated in the parameter of the function:  
Expected: A boolean value of true is still returned.  
Actual: The function returned true. (No GUI implementation)
  - c. If the balance of the object is less than the value in the parameter:  
Expected: A false value is returned by the function.  
Actual: The function returned false. (No GUI implementation)

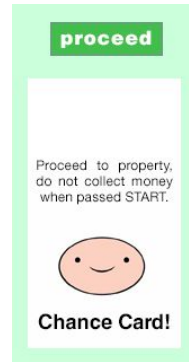
## Deck

1. Public boolean drawCard(Player, player, Board b, int loc, Bank bank)
  - a. If the sizeOfDeck is greater than 0 :  
Expected: The function obtains the Card object from the index (sizeOfDeck) and applies its conditions. With this, the sizeOfDeck is also decremented.  
Actual: Size of Deck was decremented after draw card.

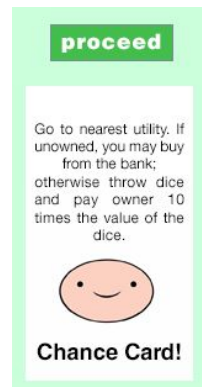


- b. If the sizeofDeck is equal to 0:  
Expected: The function shuffles the deck initialized during instantiation and recurs itself to obtain the Card instance from the last index in the array.  
Actual : The player was still able to draw card because the cards from discarded pile were shuffled.





- c. If the sizeOfDeck is full :
- Expected: The function obtains the Card object from the last index and applies its conditions. With this, the sizeOfDeck is also decremented.
- Actual: The size of deck was decremented after draw card.



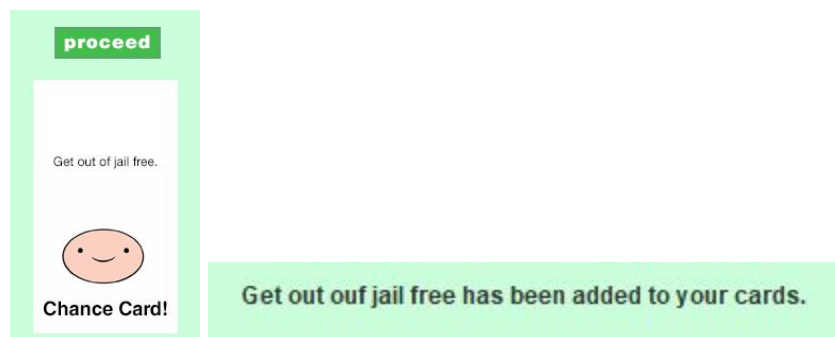
## Card

1. Public applyEffect (Player player, Board b, int loc, Bank bank)

- a. **Category1**

Expected: The card applies the effect - *Get out of jail free*. With this, the card object is stored to the chance card property of the Player if it is not currently in the location of Jail.

Actual: Get out of jail was added to player's cards.

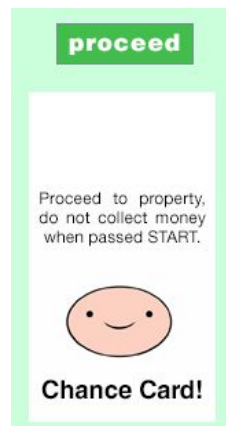


- b. **Category2A**

Expected: The card applies the effect - *Proceed to property, do not collect*

*money when passed START. You may buy the property if unowned, pay rent, or propose trade.* The function identifies the nearest property from the location of the Player that drew the card and places the instance in that location. This also considers the increments of foot traffic for each property traversed.

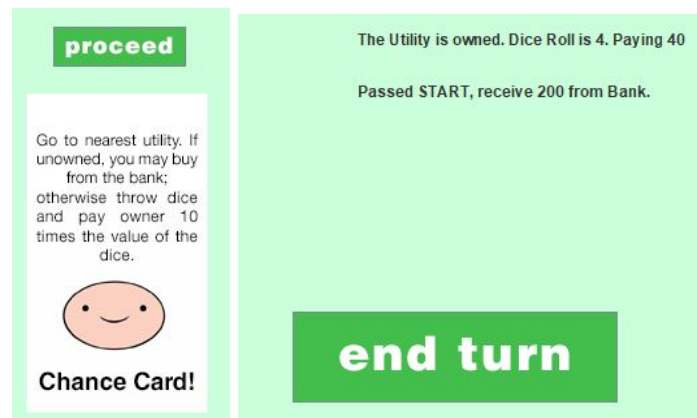
Actual: Player proceeded to property and did not collect money when passed start.



c. **Category2B**

Expected: The card applies the effect - *Go to nearest utility. If unowned, you may buy from the bank; otherwise throw dice and pay owner 10 times the value of the dice.* The function moves the Player object that drew the card to the nearest Utility Space. The foot traffic values of the Property objects traversed are also incremented.

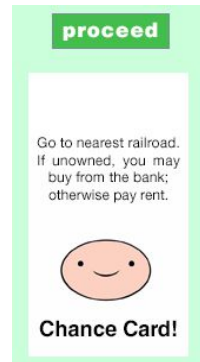
Actual: Player paid owner dice roll value times 10.



d. **Category2C**

Expected: The card applies the effect - *Go to nearest railroad. If unowned, you may buy from the bank; otherwise pay rent.* The function moves the Player to the nearest Railroad Space. Values of the foot traffic from Property objects traversed are likewise increased by 1.

Actual: Player proceeded to nearest railroad



**e. Category3A**

f. Expected: The card applies one out of the following effects

- i. *Congratulations! Bank pays dividend of \$50.*
- ii. *Tax refund. Collect \$100 from the bank.*
- iii. *It's your birthday! Collect \$300 gift money.*
- iv. *You won the competition, collect \$150 prize money.*

Actual: The card applied effect. Player collected 100 from the bank.



**g. Category3B**

Expected: The card applies the effect - *Advance to START, collect \$200.* The location of the player is changed to index 0 as the start of the Board. Foot traffic values are incremented between the Property objects between the previous location and Start.

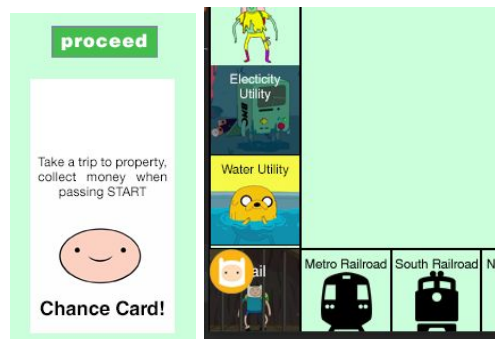
Actual: Player advanced to start and collected 200.



**h. Category4A**

Expected: The card applies the effect - *Go to jail. When passing START, do not collect \$200*

Actual: Player proceeded to jail and did not collect money when passing through start.



**i. Category4B**

Expected: The card applies the effect - *Take a trip to property, collect money when passing START.*

Actual: Player proceeded to property and collected money when passed through start.



**j. Category5A**

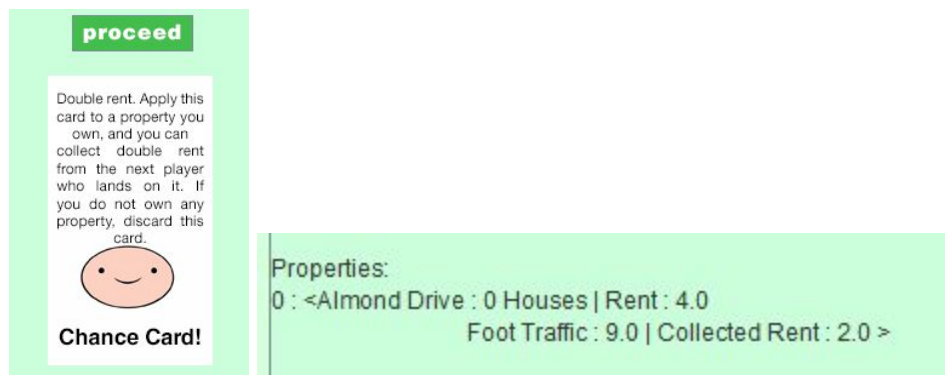
Expected: The card applies the effect - *Double rent. Apply this card to a property you own, and you can collect double rent from the next player*

who lands on it. If you do not own any property, discard this card. If player input is invalid, the program will ask again for an input.

I. Actual: The card was discarded because player does not own any property.



II. Actual: Property rent was doubled.



k. **Category5B**

Expected: The card applies the effect - *Apply this card to a property you own. Renovation costs \$25 per house, or \$50 per hotel. From now on, rent is increased by 50%. If you do not own any property, discard this card.* If player input is invalid, ask again for input. The function continuously scans for a number of the property which the effect will be applied to, given that the Player has at least 1 property.

Actual: The property chosen was renovated and player paid bank.

The screenshot shows a green-bordered area containing a 'Chance Card!' and a 'confirm' button. The card has a green header with the word 'proceed' in white. The text on the card reads: 'Apply this card to a property you own. Renovation costs \$25 per house, or \$50 per hotel. From now on, rent is increased by 50%. If you do not own any property, discard this card.' Below the text is a simple orange smiley face icon. The card is labeled 'Chance Card!' at the bottom. To the right of the card is a green box with a white input field labeled 'Enter index :'. The input field contains the number '0'. Below the input field is a green button with the word 'confirm' in white.

### l. **Category5CA**

Expected: The card applies the effect - *Apply this card to a utility or railroad you own. Increase charge by 10%. If you do not own any utility or railroad, discard this card.* If user does not own railroad and owns utility, it will automatically for the utility index and vice versa. It will ask again for an index if the input is invalid until a valid index is input.

Actual : It asked for U or R then the index, since input is valid. Rent was increased by 10%

The screenshot shows a green-bordered area containing a 'Chance Card!' and two 'confirm' buttons. The card has a green header with the word 'proceed' in white. The text on the card reads: 'Apply this card to a utility or railroad you own. Increase charge by 10%. If you do not own any utility or railroad, discard this card.' Below the text is a simple orange smiley face icon. The card is labeled 'Chance Card!' at the bottom. To the right of the card are two green boxes. The first box has a white input field labeled 'Enter U or R :'. The second box has a white input field labeled 'Enter index :'. Both input fields contain the number '0'. Below each input field is a green button with the word 'confirm' in white.

### m. **Category5CB**

Expected: The card applies the effect - *Apply this card to a utility or railroad you own. Decrease charge by 10%. If you do not own any utility or railroad, discard this card.* If user does not own railroad and owns utility, it will automatically for the utility index and vice versa. It will ask again for an index if the input is invalid until a valid index is input.

Actual: It asked for U or R then the index, since input is valid. Rent was decreased by 10%.

Newloc : Chance 2

**proceed**

Apply this card to a utility or railroad you own. Decrease charge by 10%. If you do not own any utility or railroad, discard this card.



Chance Card!

Enter U or R :

**confirm**

Enter index :

**confirm**

**n. Category6**


Expected: The card applies one out of the following effects

- Donate money for community development (random amount).* The function generates a random value to be donated that ranges from 0 to the current balance of a Player
- The card applies the effect - *Pay taxes (random amount)* - The function generates a random value to be donated that ranges from 0 to the current balance of a Player

Actual : The card applied effect and player donated random amount.

**proceed**

Donate money for community development (random amount)



Chance Card!

You paid 407.3571564953152

**end turn**

**2. Public int nearest(String type, Board b, int loc)**

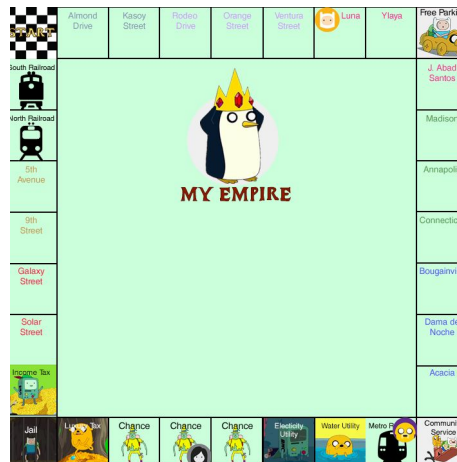
- If nearest space is before player passes start

Expected: Method will return index less than old index.

Actual: Player proceeded to space



- b. If nearest space is after player passes start  
 Expected: Method will return index greater than old index.  
 Actual: Player did not pass start.



- c. If nearest space is equal to old space :  
 Expected: Method will return index equal to old index.  
 Actual: This is not possible in the actual board setup.

## Property

1. public boolean isOwned()
  - a. If property does not have an owner  
 Expected: Function returns false  
 Actual: Function returned false



- b. If the property is owned by a player



Expected: Function returns true

Actual: Function returned true

```
Properties:  
0 : <Orange Street : 0 Houses | Rent : 16.0  
Foot Traffic : 25.0 | Collected Rent : 28.0 >
```

c. If the owner of the property changes

Expected: Function still returns true

Actual: Function returned true

```
[Player num: 3 Money : 1487.0]  
Properties : 7 Utilities : 0 Railroads : 0 Chance Cards : 0  
  
Properties:  
0 : <Rodeo Drive : 2 Houses | Rent : 100.0  
Foot Traffic : 25.0 | Collected Rent : 116.0 >
```

2. public boolean canBeDeveloped()

a. If property is not yet fully developed but collected rent is not yet enough and traffic count is not yet enough:

Expected: The method will return false.

Actual: The property was not developed



b. If property is fully developed:

Expected: The method will return false.

Actual: The method returned, false, property cannot be developed.

Ventura Street cannot be developed.

c. If the property is not yet fully developed and foot traffic count is enough:

Expected: The method will return true.

Actual: The property can be developed, function returned true.

```
1 : <Kasoy Street : 1 Houses | Rent : 20.0  
Foot Traffic : 24.0 | Collected Rent : 4.0 >
```

d. If the property is not yet fully developed and total collected rent is enough:

Expected: The method will return true.

Actual: The method returned true, property was developed.

Almond Drive was developed.