Larraquel, Reign Elaiza D. 02/07/22

S15B CCPROG

PROPERTY... PROPERTY

Funionct: int changeSettingValue(int nAnswer)

Function description:
The user inputs a new cash value that will replace the original values of the following settings: Railroad Rent, Electric Company Rent, Railroad Cost, Electric Company Cost, Renovation Cost, and Initial Cash

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player changes the value for Initial Cash	nAnswer = 6 nNewVal = 100	100	100	P
2	Player changes the value for Railroad Rent	nAnswer = 1 nNewVal = -12	[INVALID INPUT] (Ask for another user input)	[INVALID INPUT] (Ask for another user input)	P
3	Player changes the value for Renovation Cost	nAnswer = 5 nNewVal = 1	1	1	P

Function: void setLuckyRange(LuckyRange* luckyRange)

Function description:
Changes the range of amounts in Feeling' Lucky such as the maximum amount if the dice value is prime, minimum amount if the dice value is prime, maximum amount if the dice value is not prime, and minimum amount if the dice value is not prime.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	minimum amount is larger than the maximum amount	nPrimeMax = 10 nPrimeMin = 15 nNonPrimeMax = 1 nNonPrimeMin = 5	nPrimeMax = 10 [INVALID INPUT] (Ask for another user input) nNonPrimeMax = 1 [INVALID INPUT] (Ask for another user input)	nPrimeMax = 10 [INVALID INPUT] (Ask for another user input) nNonPrimeMax = 1 [INVALID INPUT] (Ask for another user input)	P
2	Maximum is less than 1	nPrimeMax = 0 nPrimeMin = 1 nNonPrimeMax = 0 nNonPrimeMin = 1	[INVALID INPUT] (Ask for another user input) [INVALID INPUT] (Ask for another user input) [INVALID INPUT] (Ask for another user input) [INVALID INPUT]	[INVALID INPUT] (Ask for another user input) [INVALID INPUT] (Ask for another user input) [INVALID INPUT] (Ask for another user input) [INVALID INPUT]	P
3	minimum is less than maximum amount	nPrimeMax = 14 nPrimeMin = 12 nNonPrimeMax = 6 nNonPrimeMin = 2	nPrimeMax = 14 nPrimeMin = 12 nNonPrimeMax = 6 nNonPrimeMin = 2	nPrimeMax = 14 nPrimeMin = 12 nNonPrimeMax = 6 nNonPrimeMin = 2	P

Function: void setCashLimit (int nInitialCash, int* nCashLimit, int* nEndCond)

Function description:
This function allows the user to activate another game end condition where the game ends when a specific cash-on-hand is reached. The specific cash-on-hand that the user must reach should be more than the initial cash to not cause the game to end instantly. The user may also choose between which condition (or both) must be satisfied.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F

1	Cash limit is less than the initial cash and the end condition that must be satisfied is if one player does not have cash and	nInitialCash = 200 nCashLimit = 199 nEndCond = 1	[INVALID INPUT] (Ask for another user input) nEndCond = 1	[INVALID INPUT] (Ask for another user input) nEndCond = 1	P
2	Cash limit is greater than the initial cash and the end condition to satisfy is the specific cash-on-hand	nInitialCash = 50 nCashLimit = 2000 nEndCond = 2	nCashLimit = 2000 nEndCond = 2	nCashLimit = 2000 nEndCond = 2	P
3	Cash limit is greater than the initial cash and the end condition to satisfy is	nInitialCash = 100 nCashLimit = 2000 nEndCond = 3	nCashLimit = 2000 nEndCond = 3	nCashLimit = 2000 nEndCond = 3	P

Function: void settings(int nAnswer, int* nInitialCash, Rent* rent, Cost* cost, LuckyRange* luckyRange, int* nCashLimit, int* nEndCond)

Function description:
This function will call another function depending on what the user plans to change specifically and it will replace the initial value of the setting to what the user inputs.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player changes the value for Initial Cash	nAnswer = 6 nInitialCash = 100	nInitialCash = 100	nInitialCash = 100	P
2	Player changes the value for Railroad Rent	nAnswer = 1 nRailRent = -12	[INVALID INPUT] (Ask for another user input)	[INVALID INPUT] (Ask for another user input)	P
3	Player changes the value for Renovation Cost	nAnswer = 5 nRenovCost = 1	nRenovCost = 1	nRenovCost = 1	P

Function: void menu(int* nGame, int* nInitialCash, Rent* rent, Cost* cost, LuckyRange* luckyRange, int* nCashLimit, int* nEndCond)

Function description:
This function presents the users with options in a form of a menu where the user may choose to start a new game, change the settings, or exit the program.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player chooses to start a new game	nGame = 1	nGame = 1	nGame = 1	P
2	Player chooses to edit the settings	nGame = 2	(Display Settings)	(Display Settings)	P
3	Player chooses to exit the program	nGame = 3	nGame = 3	nGame = 3	P

Function: int rollDice()

Function description:
Rolls the dice in the range of 1 to 6

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player rolled the dice	N/A	No Expected Result as it is random	6	P
2	Player rolled the dice	N/A	No Expected Result as it is random	3	P
3	Player rolled the dice	N/A	No Expected Result as it is random	5	P

Function: char positionType(int nPosition)

Function description:
Returns a character that determines the type of the position the player landed on

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player landed on a property	nPosition = 3	P	P	P
2	Player landed on Jail	nPosition = 4	J	J	P
3	Player landed on the Railroad	nPosition = 7	R	R	P

Function: int getProperty(int nProperties, int nPosition)

Function description:
Extracts the status of the needed property according to the position the player lands.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Property is unowned	nProperties = 120 nPosition = 1	0	0	P
2	Property is a unrenovated property of player 1	nProperties = 310030120 nPosition = 3	1	1	P
3	Property is a renovated property of player 2	nProperties = 310040120 nPosition = 5	4	4	P

Function: int checkProperties(int nProperties, int nPlayer)

Function description:
Checks if the player has any properties owned by them

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 has no properties	nProperties = 0 nPlayer = 1	0	0	P
2	Player 2 has only unrenovated properties	nProperties = 2000202 nPlayer = 2	1	1	P
3	Player 2 has both unrenovated and renovated properties	nProperties = 23040102 nPlayer = 2	1	1	P

Function: int checkRenovation(int nProperties, int nPosition, int nPlayer)

Function description:
Checks if a player's owned house property is renovated or not.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 2 has no properties	nProperties = 0 nPosition = 1 nPlayer = 2	0	0	P
2	Player 2 has only unrenovated properties	nProperties = 30030120 nPosition = 2 nPlayer = 2	1	1	P
3	Player 2 has renovated properties	nProperties = 40040120 nPosition = 8 nPlayer = 2	2	2	P

Function: int getPropertyCost(int nPosition, Cost cost)

Function description:
Gets the cost of the property according its type

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	The property is a house property	nPosition = 1	20	20	P
2	The property is the electric company	nPosition = 2 nElecCost = 200	200	200	P
3	the property is the railroad	nPosition = 7 nRailCost = 100	100	100	P

Function: int getPropertyRent(int nPlayer, int nPosition, int nProperties, int nDice, Rent rent, Cost cost)

Function description:
Gets the rent of the property according its type

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	The property is a renovated house property	nPlayer = 1 nPosition = 3 nProperties = 412 nDice = 3	25	25	P
2	The property is the electric company	nPlayer = 2 nPosition = 2 nProperties = 412 nDice = 3 nElecRent = 8	24	24	P
3	the property is the railroad	nPlayer = 2 nPosition = 7 nProperties = 1000412 nDice = 3 nRailRent = 30	30	30	P

Function: int checkOwnership(int nProperties, int nPlayer, int nPosition)

Function description:
Checks if the player owns the property that is inputted

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 owns the unrenovated property	nProperties = 201 nPlayer = 1 nPosition = 1	1	1	P
2	Player 1 owns the renovated property	nProperties = 30201 nPlayer = 1 nPosition = 5	1	1	P
3	Player 1 does not own the property	"nProperties = 201 nPlayer = 1 nPosition = 3	0	0	P

Function: void changeOwnership(int* nProperties, int nPosition, int nPlayer, int nSign)

Function description:
Edits the values of the 9 digit integer property

#	Test Description	Sample Input	Expected Result	Actual Result	P/F

1	Player 1 sells a property	nProperties = 20030211 nPosition = 2 nPlayer = 1 nSign = 1	nProperties = 20030201	nProperties = 20030201	P
2	Player 1 buys a property	nProperties = 20030211 nPosition = 7 nPlayer = 1 nSign = 2	nProperties = 21030211	nProperties = 21030211	P
3	Player 1 renovates a property	nProperties = 20030211 nPosition = 1 nPlayer = 1 nSign = 3	nProperties = 20030213	nProperties = 20030213	P

Function: void displayPosition(int nDice, int* nPosition, int* nCash)

Function description:
Moves the player to their new position depending on the dice roll

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player rolled 6 on the dice while in position 3	nDice = 6 nPosition = 3 nCash = 200	nPosition = 9 nCash = 200	nPosition = 9 nCash = 200	P
2	Player rolled 3 on the dice while in position 9	nDice = 3 nPosition = 9 nCash = 200	nPosition = 2 nCash = 400	nPosition = 2 nCash = 400	P
3	Player rolled 2 on the dice while in position 8	nDice = 2 nPosition = 8 nCash = 100	nPosition = 0 nCash = 300	nPosition = 0 nCash = 300	P

Function: int isEnoughCash(int nCash, int nCost)

Function description:
Checks if the player has enough cash to pay for the cost

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player has more cash than what is needed to pay	nCash = 200 nCost = 50	1	1	P
2	Player has less cash than what is needed to pay	nCash = 50 nCost = 200	0	0	P
3	Player has the same amount of cash than what is needed to pay	nCash = 50 nCost = 50	1	1	P

Function: void cashPay(int* nCash, int nCost)

Function description:
Displays a message if the player's cash is enough to pay for the cost and subtracts the cost from the player's cash

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player has more cash than what is needed to pay	nCash = 200 nCost = 50	nCash = 150	nCash = 150	P
2	Player has less cash than what is needed to pay	nCash = 50 nCost = 200	Insufficient Cash	Insufficient Cash	P
3	Player has the same amount of cash than what is needed to pay	nCash = 50 nCost = 50	nCash = 0	nCash = 0	P

Function: void jail(Player* player1, Player* player2, int nPlayer)

Function description:
Places the player in jail when they land on the position 4 (Jail Time). When in jail, the player lose their next turn

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 rolls a 1 in Feelin Lucky	nPlayer = 1	player1.nJail = 1	player1.nJail = 1	P
2	Player 2 lands in position 4	nPlayer = 2	player2.nJail = 1	player2.nJail = 1	P
3	Player 1 lands in position 4	nPlayer = 1	player1.nJail = 1	player1.nJail = 1	P

Function: void resellProperty(int nPosition, int nPlayer, int* nProperties, int *nCash, Cost cost)

Function description:
This function allows the user to resell a property to the bank and updates the status of the ownership by changing the value of the property that was sold to 0. The resell value is half the cost of the property

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 2 sells a renovated house property	nPosition = 1 nPlayer = 2 nProperties = 22040224 nCash = 200	nProperties = 22040220 nCash = 220	nProperties = 22040220 nCash = 220	P
2	Player 2 sells the electric company	nPosition = 2 nPlayer = 2 nProperties = 22040224 nCash = 200 nElecCost = 150	nProperties = 22040204 nCash = 275	nProperties = 22040204 nCash = 275	P
3	Player 2 sells the railroad	nPosition = 7 nPlayer = 2 nProperties = 22040224 nCash = 200 nRailCost = 100	nProperties = 20040224 nCash = 250	nProperties = 20040224 nCash = 250	P

Function: int isPrime(int nDice)

Function description:
Checks if the dice value is a prime number or not

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Dice roll is 5	nDice = 5	1	1	P
2	Dice roll is 3	nDice = 3	1	1	P
3	Dice roll is 4	nDice = 4	0	0	P

Function: void feelingLucky(int* nCash, Player* player1, Player* player2, LuckyRange luckyRange, Cost cost, int* nPosition, int nPlayer, int* nProperties)

Function description:
Determines the action that happens when the player lands in Position 6 (Feelin' Lucky)

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Dice roll is a prime number	nCash = 200 nPosition = 6 nPlayer = 2 nDice = 2 nPrimeMax = 200 nPrimeMin = 100	(Number in the range of 100 - 200)	nCash = 325	P

2	Dice roll is NOT a prime number	nCash = 200 nPosition = 6 nPlayer = 2 nDice = 4 nNonPrimeMax = 200 nNonPrimeMin = 100	(Number in the range of 100 - 200)	nCash = 68	P
3	Dice roll is 1	nCash = 200 nPosition = 6 nPlayer = 2 nDice = 1 nProperties = 201	player2.nJail = 1 nPosition = 4	player2.nJail = 1 nPosition = 4	P

Function: void buyProperty(int* nCash, int* nProperties, int nPosition, int nPlayer, Cost cost)

Function description:
If the player decides to buy the property, this subtracts the property's cost from the player's cash and adds the property to the player's ownership.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 buys a house property	nCash = 200 nProperties = 30000100 nPosition = 1 nPlayer = 1 nAnswer = 1	nProperties = 30000101 nCash = 180	nProperties = 30000101 nCash = 180	P
2	Player 1 buys the Electric Company	nCash = 200 nProperties = 30000100 nPosition = 2 nPlayer = 1 nElecCost = 100	nProperties = 30000011 nCash = 100	nProperties = 30000011 nCash = 100	P
3	Player 1 buys the Railroad	nCash = 200 nProperties = 30000100 nPosition = 7 nPlayer = 1 nRailCost = 50	nProperties = 31000001 nCash = 150	nProperties = 31000001 nCash = 150	P

Function: void payRent(int* nCash, int* nProperties, Player* player1, Player* player2, int nPosition, int nDice, int nPlayer, Rent rent, Cost cost)

Function description:
The player has to pay the other player the rent cost for landing in their owned property.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 landed in a renovated house property owned by another player	nCash = 200 nProperties = 412 nPosition = 3 nDice = 3 nPlayer = 1	nCash = 175 player2.nCash = 25	nCash = 175 player2.nCash = 25	P
2	Player 2 landed in the electric company owned by another player	nCash = 200 nProperties = 412 nPosition = 2 nDice = 3 nPlayer = 2 nElecRent = 8	nCash = 176 player2.nCash = 24	nCash = 176 player2.nCash = 24	P

3	Player 2 landed in the Railroad owned by another player	nCash = 200 nProperties = 1000412 nPosition = 7 nDice = 3 nPlayer = 2	nCash = 170 player2.nCash = 30	nCash = 170 player2.nCash = 30	P

Function: void renovateProperty(int* nCash, int* nProperties, int nPosition, int nPlayer, Cost cost)

Function description:
If they choose to renovate, the rent value of the property increases but the player would need to pay the renovation cost.

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 landed on their unrenovated property	nCash = 200 nProperties = 201 nPosition = 1 nPlayer = 1 nAnswer = 1	nCash = 170 nProperties = 203	nCash = 170 nProperties = 203	P
2	Player 2 landed on their unrenovated property	nCash = 200 nProperties = 201 nPosition = 3 nPlayer = 2 nAnswer = 1	nCash = 150 nProperties = 401	nCash = 150 nProperties = 401	P
3	Player 1 landed on their unrenovated property	nCash = 200 nProperties = 201 nPosition = 1 nPlayer = 1 nAnswer = 2	nCash = 200 nProperties = 201	nCash = 200 nProperties = 201	P

Function: void changePlayer(int* nCash, int* nPosition, int* nPlayer, Player* player1, Player* player2)

Function description:
Stores the information of the player in the struct Player and changes the current player to the next player

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 is the current player and player 2 is not in jail	nCash = 300 nPosition = 4 nPlayer = 1 player1.nCash = 200 player1, nPosition = 2 player2.nCash = 200 player2, nPosition = 5 player1.nJail = 0 player2.nJail = 0 player1.nPlayer = 1	player1.nCash = 300 nCash = 200 player1.nPosition = 4 nPosition = 5 nPlayer = 2	player1.nCash = 300 nCash = 200 player1.nPosition = 4 nPosition = 5 nPlayer = 2	P
2	Player 1 is the current player and player 2 is in jail	nCash = 320 nPosition = 7 nPlayer = 2 player1.nCash = 200 player1, nPosition = 2 player2.nCash = 200 player2, nPosition = 5 player1.nJail = 0 player2.nJail = 1 player1.nPlayer = 1	player2.nJail = 0	player2.nJail = 0	P

3	Both players are in jail	nCash = 320 nPosition = 7 nPlayer = 2 player1.nCash = 200 player1, nPosition = 2 player2.nCash = 200 player2, nPosition = 5 player1.nJail = 1 player2.nJail = 1 player1.nPlayer = 1	player1.nJail = 0 player2.nJail = 0 player1.nCash = 300 nCash = 200 player1.nPosition = 4 nPosition = 5 nPlayer = 2	player1.nJail = 0 player2.nJail = 0 player1.nCash = 300 nCash = 200 player1.nPosition = 4 nPosition = 5 nPlayer = 2	P

Function: int checkEndCondition(int nCash, int nProperties, int nPlayer, int nEndCond, int nCashLimit)

Function description:
Determines if the game is over once a player has won or lost

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Player 1 is bankrupt and the end condition is 1	nCash = -12 nProperties = 20 nPlayer = 1 nEndCond = 1 nCashLimit = 0	1	1	P
2	Player 2 has reached the specific cash on hand and the end condition is 2	nCash = 2022 nProperties = 120 nPlayer = 2 nEndCond = 2 nCashLimit = 2000	1	1	P
3	Player 1 has properties and cash	nCash = 200 nProperties = 120 nPlayer = 1 nEndCond = 3 nCashLimit = 2000	0	0	P