The Final Report – Design, Implementation and Process

Group 45

Authored by: Adam Logan

XXXXX

XXXXX

XXXXX

Queen's University Belfast

Technopoly

Contents

Peer Assessment Form

Evaluation

Evaluation	Group Number: 45			
Name	Contribution to team-working and motivation[1]	Contribution to PDF Report 2, and the Video Demo	Contribution to methodically developed and functioning system code	**Peer Score** (Range 85 – 115)
Adam Logan	5	5	5	115
XXXXX	5	4	3	105
XXXXX	5	4	2	100
XXXXX	4	4	3	105

Values for contribution: 1 = Minimal Contribution; 2 = Reasonable Contribution; 3 = Good Contribution; 4 = Very Good Contribution; 5 = Excellent Contribution

Declaration

Declaration

"I declare that I have read the Queen's University regulations on plagiarism, and that any contribution I have made to the attached submission is my own original work, except for any elements that I have clearly attributed to third parties. I understand that this submission will be subject to an electronic test for plagiarism and will also be subject to the University's regulations concerning late submission if it is received after the deadline."

Name	Date	Confirmation *(use the words shown in the example below!)*
Adam Logan	25/03/2022	I agree to the terms of the declaration
XXXXX	25/03/2022	I agree to the terms of the declaration
XXXXX	25/03/2022	I agree to the terms of the declaration
XXXXX	25/03/2022	I agree to the terms of the declaration

Personal Statements

Personal statement of XXXXX:	
The following were my most significant contributions to the Semester 2 Deliverable (100 words or less):	
Wrote the program for incomeEvent, validOptions and printInstructions. Completed the testing plans for the use cases Players Turn, Passing Go, Buy Product, Player Eliminated, and Winning. Completed Game Design documents with XXXXX and worked on the Development Process.	

Personal statement of Adam Logan:	
The following were my most significant contributions to the Semester 2 Deliverable (100 words or less):	
All use case realisations/sequence diagrams, class diagram, updated use case diagram and descriptions. I also completed the minutes. I assisted in developing request country, product bidding and in completing the security considerations and development process documentation. The features I implemented myself are building a warehouse, saving, and reloading the game, buying and charging for use of a product, eliminating a player, transferring products and the stock exchange. I also developed all the JUnit tests. I tested the money in, money out, choice to play game, stock exchange game, loss stock exchange, anti-trust, save game, restore game and close game use cases.	

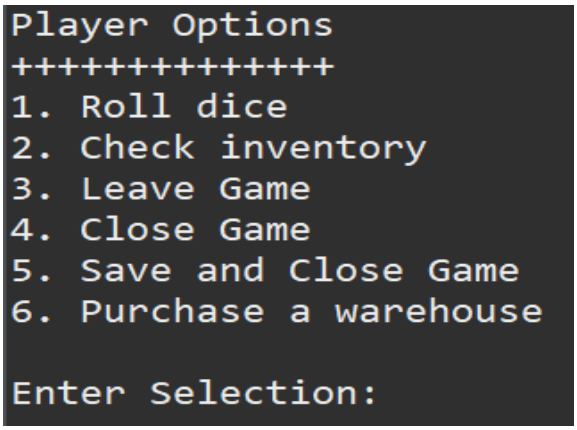
Personal statement of XXXXX:	
The following were my most significant contributions to the Semester 2 Deliverable (100 words or less):	
Testing plans for the use cases charged product, request warehouse, build warehouse. Important methods completed: productBidding, rollDice. The recording and editing video demo of our program. Weekly minutes report for the week of 21/03/2022. Additional work to Security and the Game Design documents.	

Personal statement of XXXXX:	
The following were my most significant contributions to the Semester 2 Deliverable (100 words or less):	
Completed 15 use case tests (with evidence) for the program. Also completed parts of programming along with other members (Throughout the semester, my programming software wasn't working). Test user interface also was completed.	

Text User Interface

When designing the game, there was a lot of thought put into the design, layout and then user interface, to make the game more user-friendly. Within this documentation, there will be a list of examples regarding the interface and layout, with explanations of why they were used / designed that way.

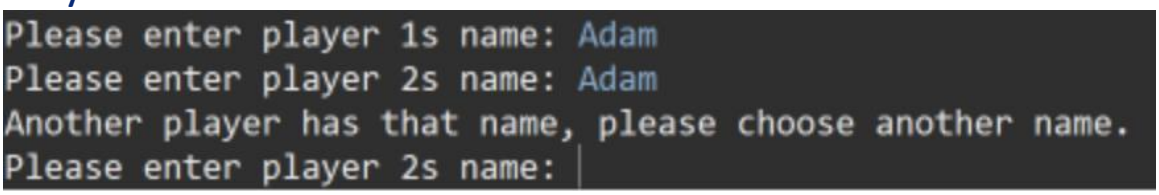
Player options menu



For the player options menu, it is designed to allow the user to easily navigate through the game. The menu box is very user-friendly and clear for all users to read and understand, making the game experience immensely better. The user would simply select which option they want and enter the corresponding number.

Another design example that we implemented was the idea that some options wouldn't be visible to users if certain requirements were not met beforehand. For example, the 'Purchase a warehouse' option (number 6), would only show to the user if they owned all the products within a specific company. This enabled the game to have validation and not allow users to 'cheat' through the game. This is also the case for a user requesting a warehouse.

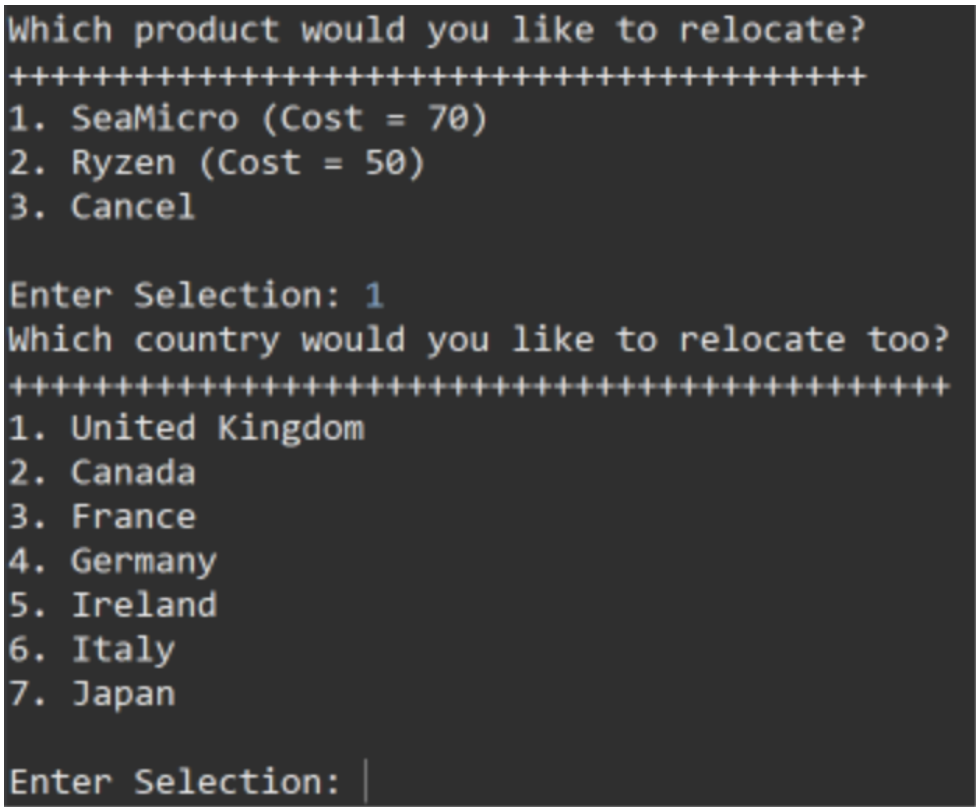
Players names



The 'players names' section is a basic, but very effective way of players entering their names. The main reason for using this form of layout is that it's very clear, and very easy for the players to follow along to

In the image above, it clearly shows that there is a form of validation within the naming process. This was done so that 2 players can't have the exact same name and confuse the system. The system then enables the user to re-enter a different name to allow them to proceed into the game

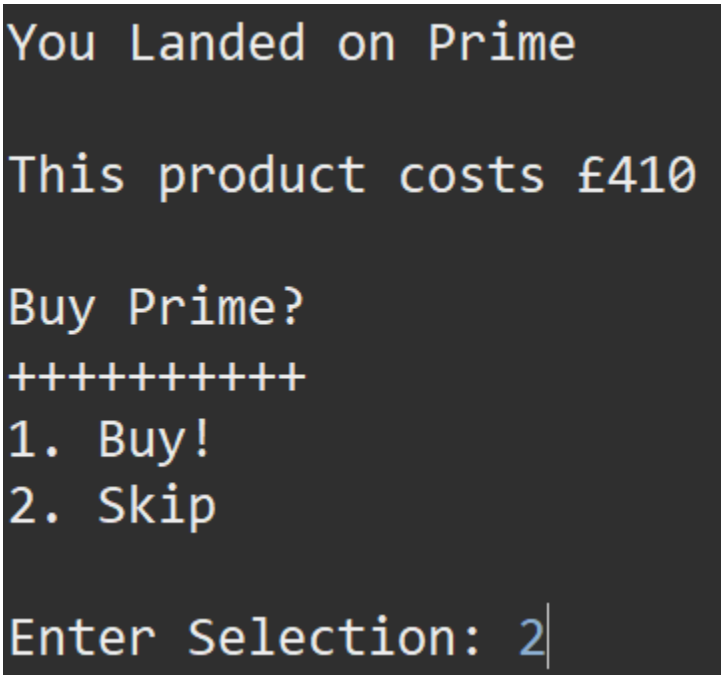
‘Purchase a county’ option



For the ‘Purchase a country’ menu option, it is a clear, user-friendly layout for the players to use. It gives clear instructions on what to do and how to use the menu in the correct way. Just like the ‘Player options menu’, The user would simply select the number corresponding to the option they would like, and the game would proceed. For the top part of the image above, the system will only be able to display products that the user owns. The reasoning for this is because it ensures that the user won’t be able to accidentally break the game and purchase a country for the wrong product. For this, we used a form of validation to ensure that this wouldn’t happen.

For the bottom part of the image above, validation is used to ensure the player can’t choose 2 of the same counties again. For example, if the player purchases ‘France’, then the next time the player goes to relocate to a country, the ‘France’ option will be removed from the list for other products in the company. The main reason for this is that it prevents the user from reusing the same country again, within the same company.

Buying Products



During the buying products stage, the layout used was chosen due to it being very clear and easy to read for the players. The system clearly displays the price of the product for the player and allows them to decide if they wish to purchase the product or not. If the player does not have the required funds to purchase the product, then the player is met with an error message. This is because of the validation that was implemented into the 'buying product' scene. validation helps to keep the game working as intended and enables the player to not accidentally eliminate themselves from the game. To make the game more difficult we decided to obscure the details of the product.

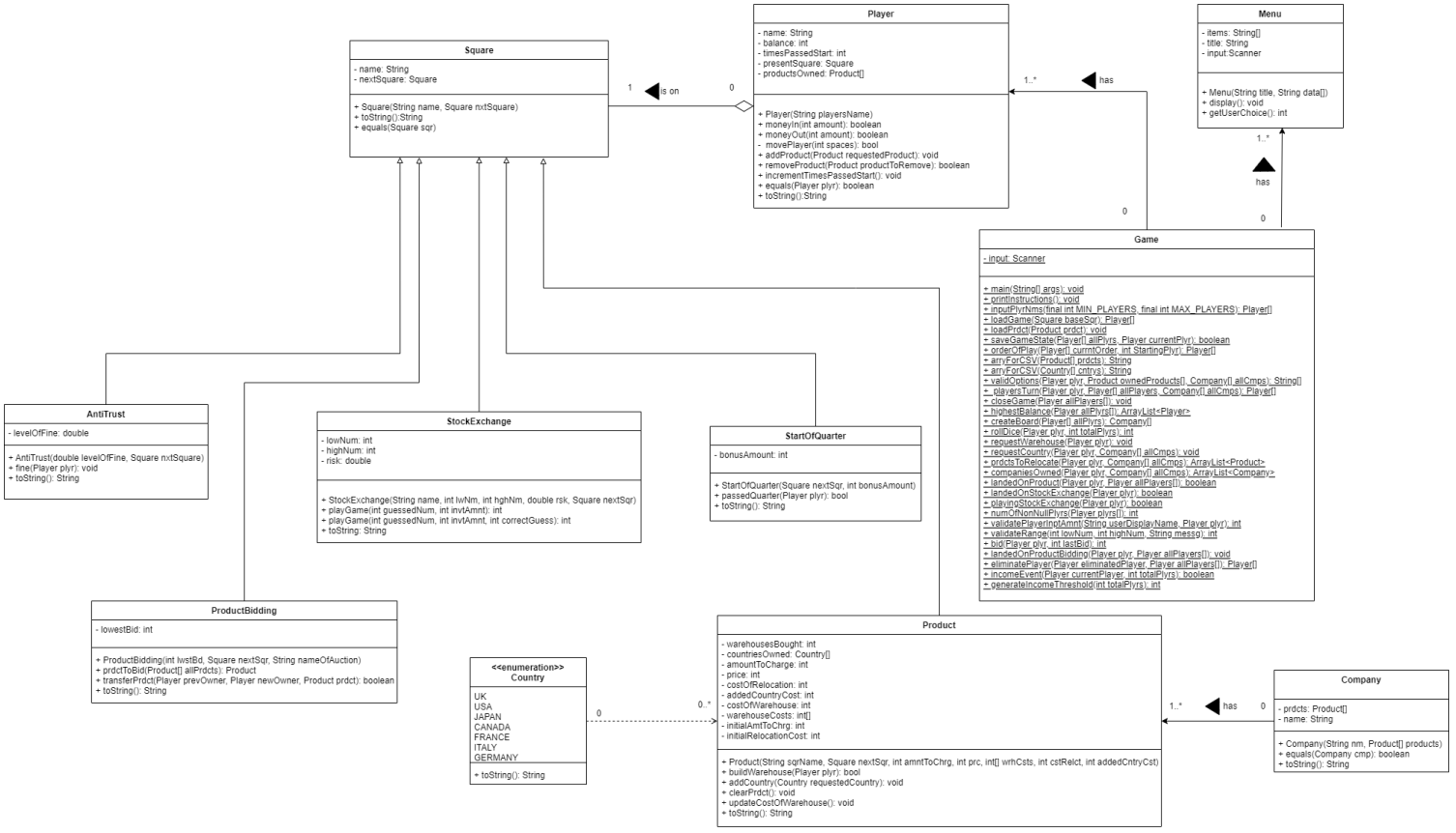
Save game file

The 'save game' function, within the game, is the first thing the players see when the game is started. from the image above, it asks the player if they wish to load a previous save from the game. The main reason for implementing this feature is to enable the users to save the game and log off the game if needs be, meaning they have no need to to play the game in one sitting. This feature is only available to the players if they have a previous game saved on their system. If the system doesn't recognise any game files on the players' system, then the feature simply will not be displayed, and the players will be brought straight to the next stage of the game.

Design Documentation

Class Diagram [A.L]

The class diagram itself [A.L]



Description of class diagram [A.L]

The class diagram above contains the candidate classes and their relationships for the system. **The getters and setters are not shown in the above diagram, but all the classes have both getters and setters for all their attributes.**

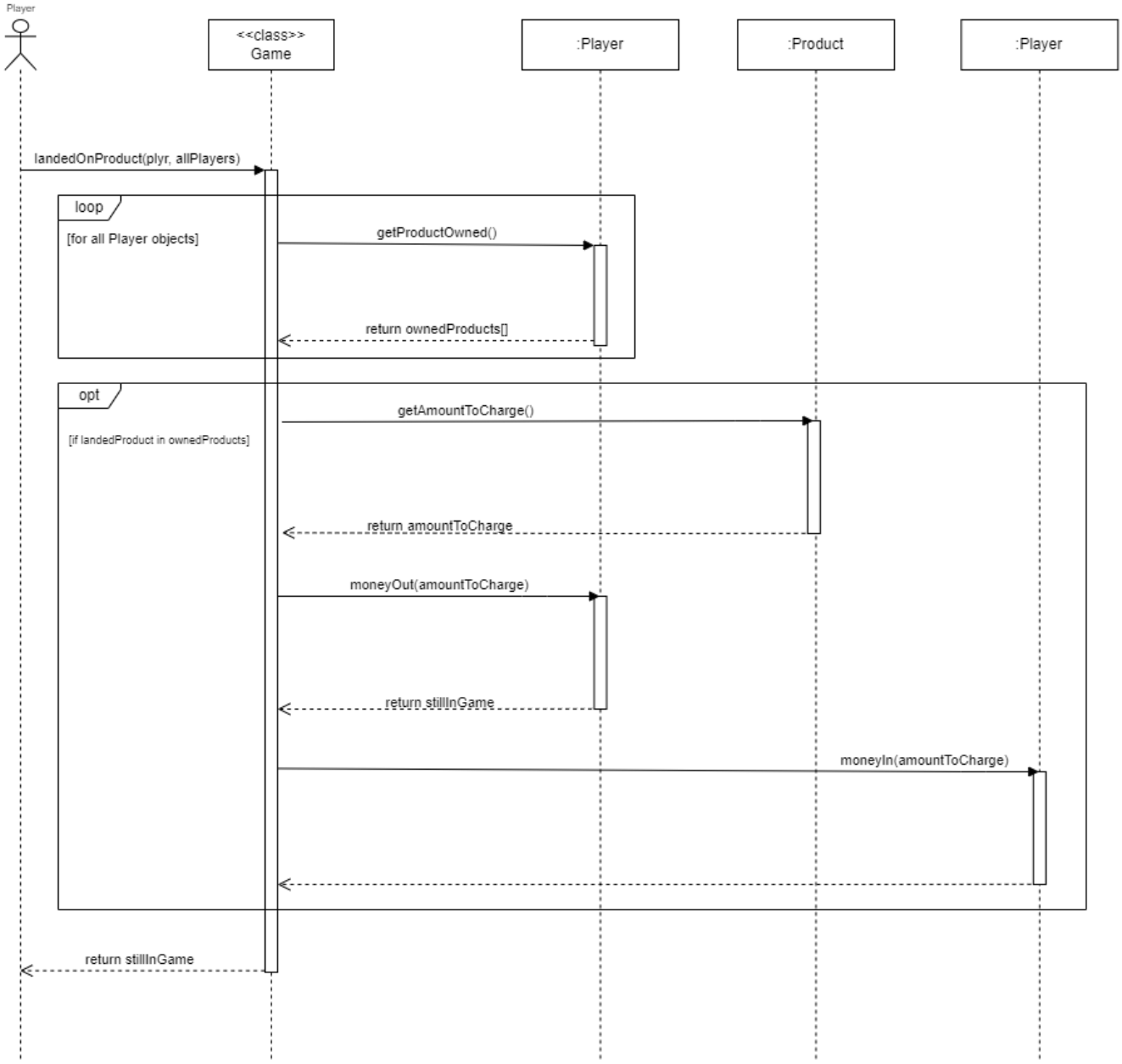
The Game class is the class which contains the 'main' method and is how the user/player interacts with the system. The Menu class is simply an easy way to create a menu for the user/player with validation. The Menu class was taken from a module in level 1, which all members of the team had enrolled.

One key change to the class diagram from the first report is the fact that the that the enumeration 'Company' is now a class. The reason for this change was that it would allow a single object to be checked to see if the Player owns all Products in the Company. The class diagram has also been updated with the extra attributes and methods that have been added throughout the project. The relationships of the classes have not changed from the first report.

Use Case Realisations [A.L]

Charged Product [A.L]

Sequence Diagram

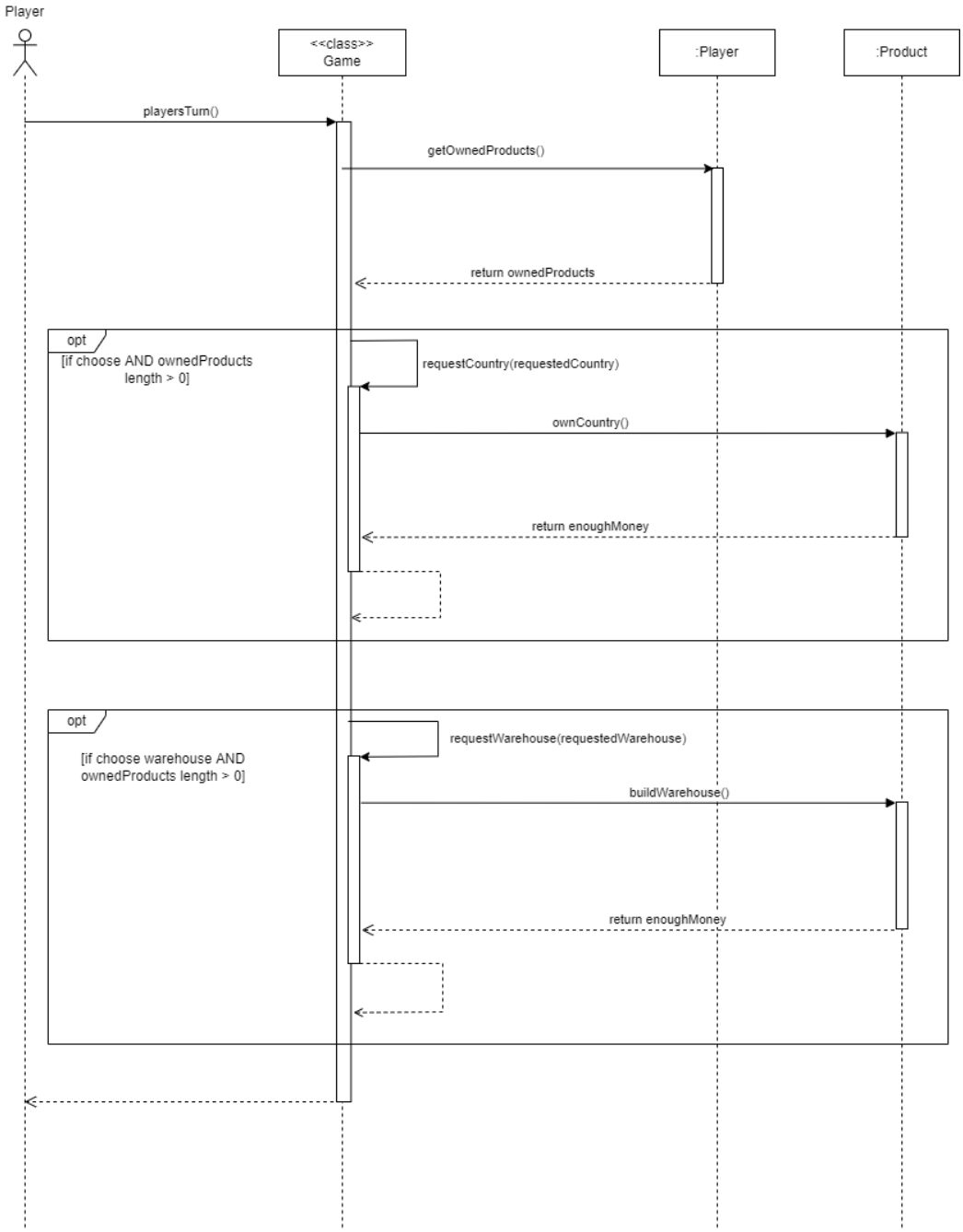


Description

This is the sequence diagram for the use cases ‘Charged Product’ which includes the ‘Money Out’ use case and the ‘Money In’ use case. This is a sequence diagram that was not included in the first report. The system gets all the products owned by all players within the game. The next step is to check if the product which was landed is owned by a player which is not the player who landed on the product. If this is the case, then the player is charged to use the product. If the player does not have the funds to pay this, then it returns ‘false’. No matter if the player who landed on the square has the funds to pay, the player who owns the product will receive the full amount.

Player's Turn [A.L]

Sequence Diagram

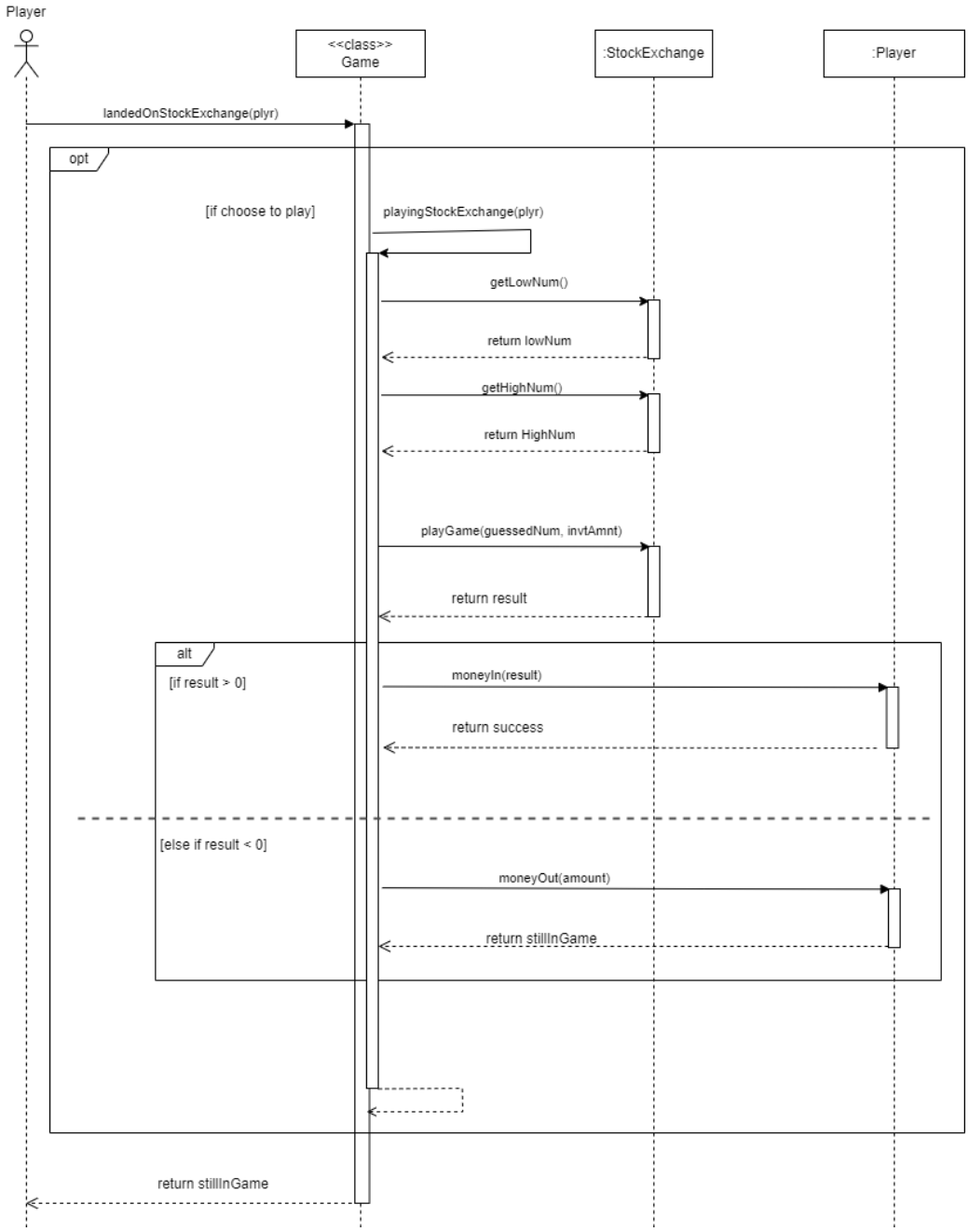


Description

This is the sequence diagram for the use cases 'Player's Turn', 'Request to buy country', 'Player Buys Country', 'Request to buy warehouse' and 'System builds warehouse'. This sequence diagram was in the first report, but changes has been made during the course of development. The key change made was, that if the player does not meet the requirements to dedicate a warehouse or relocate a country, they are simply not given the option to do this, and it I represented in the sequence diagram with the second condition in the option blocks.

Stock Exchange [A.L]

Sequence Diagram

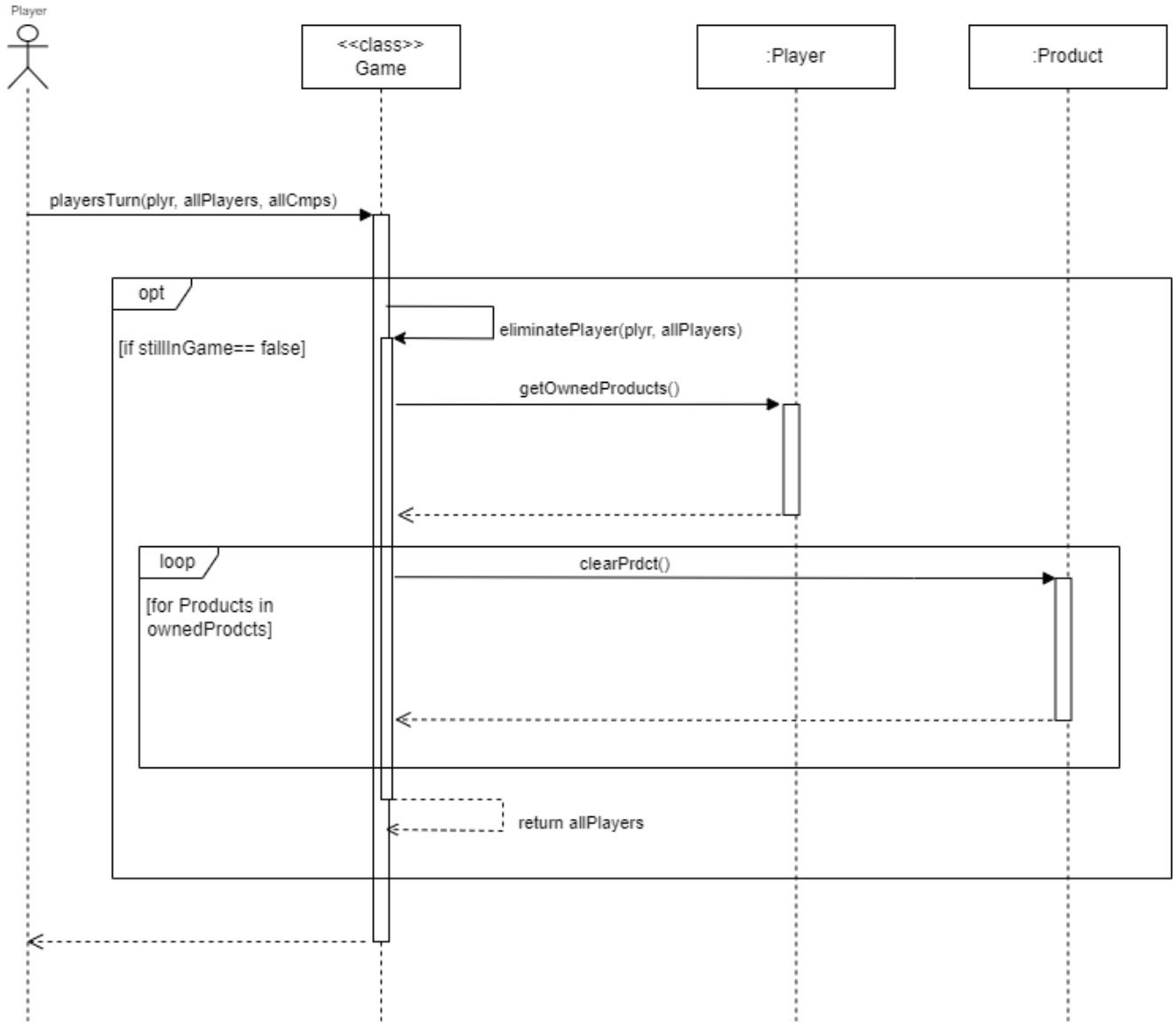


Description

This is the sequence diagram for the use cases ‘Choice to Play Game’, ‘Stock Exchange Game’, ‘Money In’ ‘Lost Stock Exchange’, ‘Money Out’. This sequence diagram was in the first report and is included here as there has been a few updates. One of these is the removal of the ‘Player Eliminated’ use case as this has gotten more complex. Another change that has been made is that there is a ‘playingStockExchange()’ method which was added to implement the ‘Choice to Play Game’ stock exchange game and to make the code more readable for maintenance. The conditions for the if and else if statements have changed, as the return type for the method ‘playGame()’. Another change has been that the calls to the methods ‘moneyIn()’ and ‘moneyOut()’ have been moved to the ‘playingStockExchange()’ method.

Eliminate Player [A.L]

Sequence Diagram

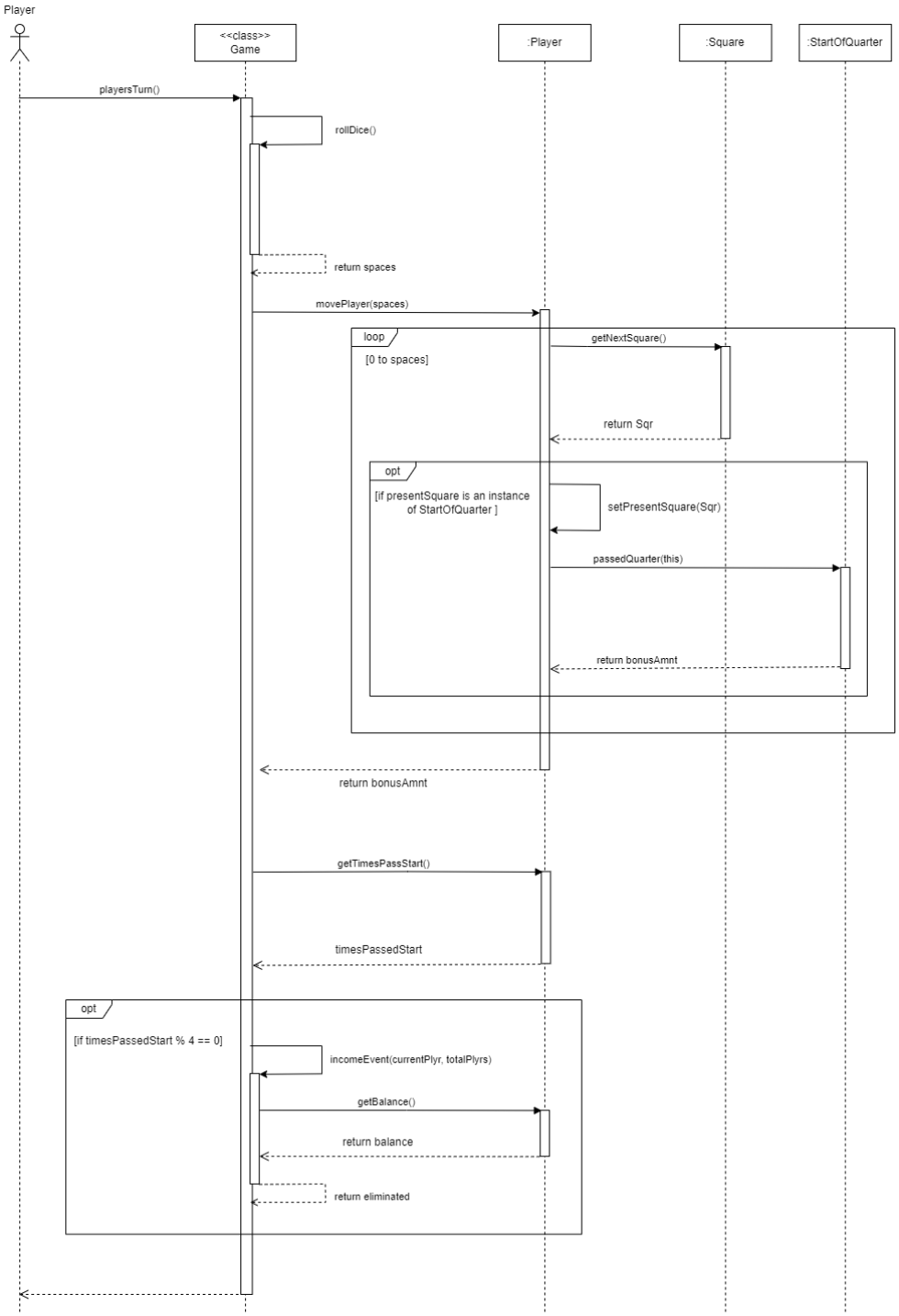


Description

This is the sequence diagram for the use case ‘Player Eliminated’. This was represented in other use case realisations within the first report, but it has gotten more complex and therefore has been extracted into its own use case realization. The main change to this has been the addition of the ‘clearPrdct()’ method, which returns the product object back to its original values. This method is looped for all the products that the player being eliminated owns. The reason why this method is called is so the remaining players can purchase these products in their original state.

Income Event [A.L]

Sequence Diagram

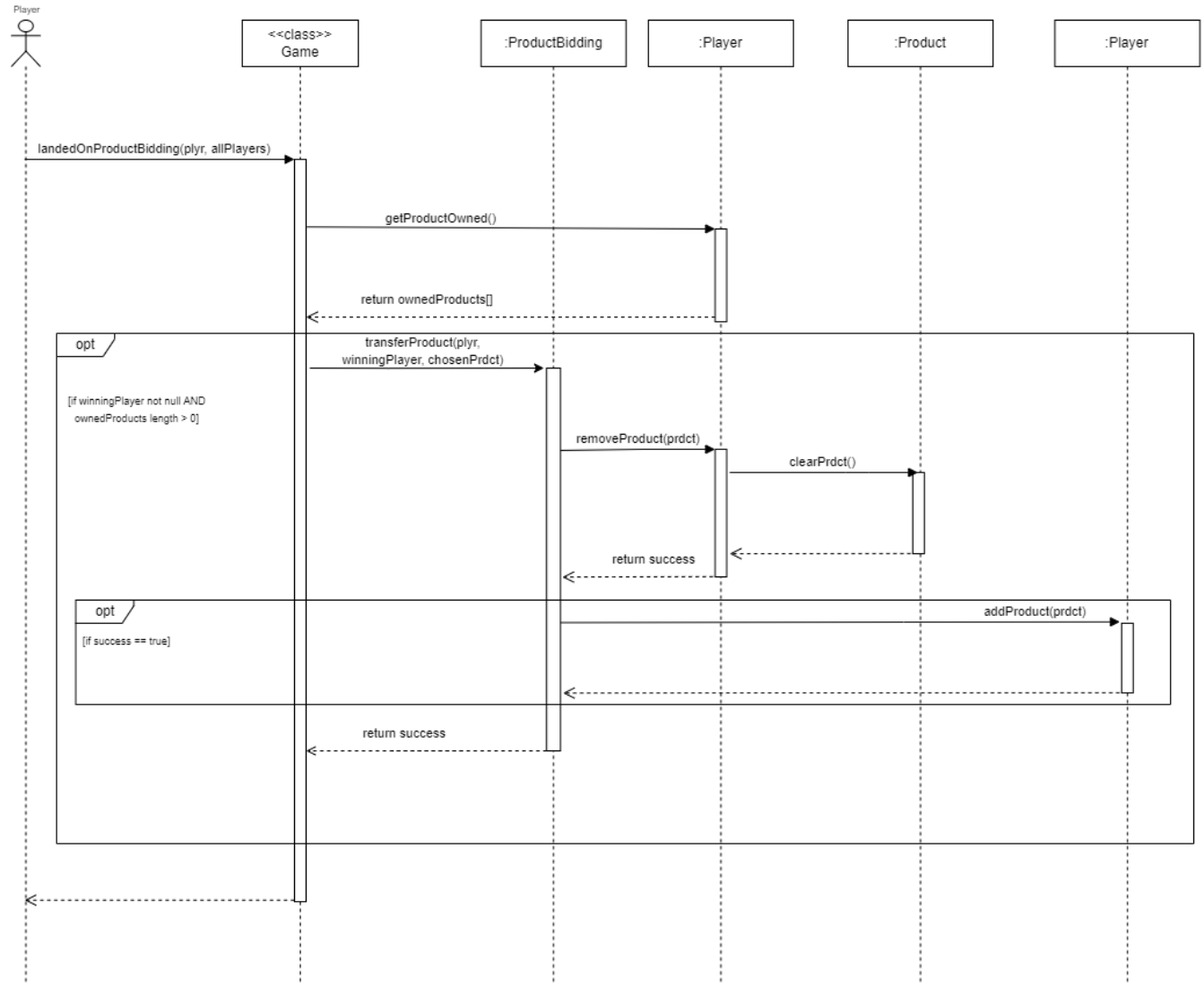


Description

This is the sequence diagram for the use cases ‘Passing Go’ and ‘Income Event’. The main change with this use case realisation is that the ‘incomeEvent()’ method has been moved to the ‘Game’ class rather than the ‘StartOfQuarter’ class. The reason for this as the new limit needs to be displayed to the user’s and therefore was better suited in this class.

Auction Starts [A.L]

Sequence Diagram

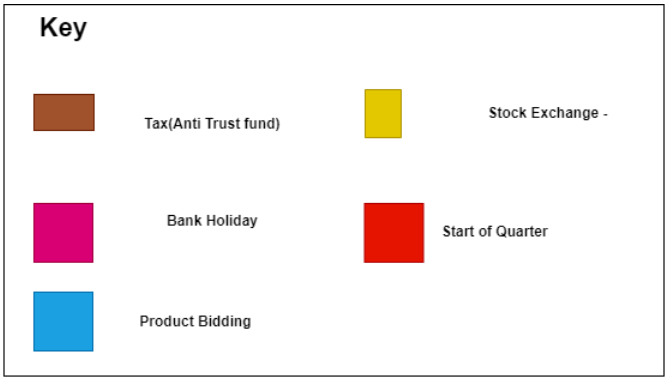


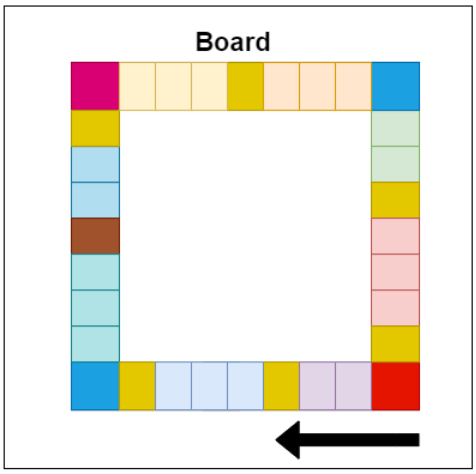
Description

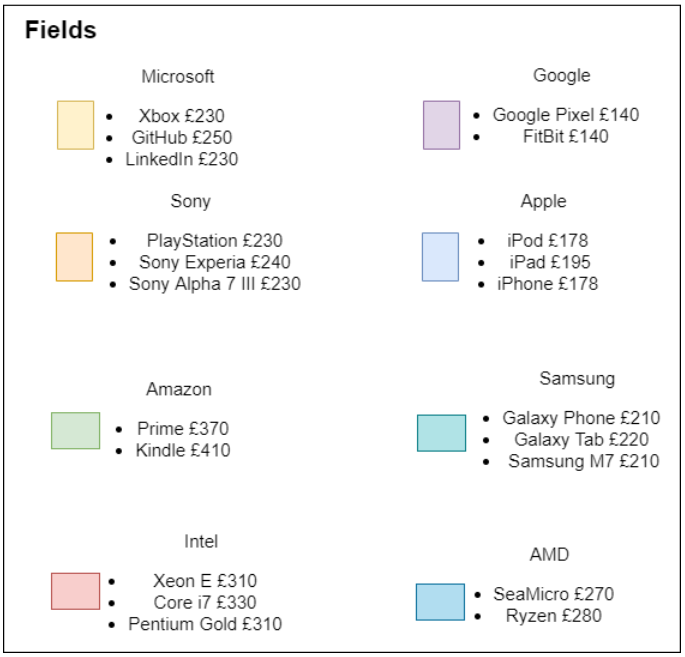
This is the sequence diagram for the use cases ‘Auction Starts’, ‘Money Out’ and ‘No Products Owned’. The differences between this use case realization and the one within the original report is the addition of the ‘transferProduct()’ and ‘clearPrdct()’ methods . Another difference is that if the player does not own any products, it no longer moves the player to a different square. The ‘clearPrdct()’ method is used here so that the winning player gets the original product rather than the enhanced version of the product which was previously owned by another player. This use case realization does not show the bidding process, but only what happens when a player wins the bid. The reason for this is to make the diagram easier to read.

Draft Game Layout [X.X]

The Board [X.X]







The Amount to Charge For Each Product [X.X]

Name	Basic	1 Warehouses	2 Warehouses	3 Warehouses	4 Warehouses	Added Country Value
Google Pixel	5	25	75	225	400	45
FitBit	5	25	75	225	400	35
iPod	7	35	105	315	460	50
iPad	9	45	135	405	595	70
iPhone	7	35	105	315	460	50
Galaxy Phone	10	50	150	450	600	295
Galaxy Tab	12	60	180	540	720	220
Samsung M7	10	50	150	450	600	295
SeaMicro	13	65	195	520	740	150
Ryzen	14	70	210	560	800	180
Xbox	15	75	225	575	825	125
GitHub	18	90	270	660	990	300
LinkedIn	15	75	225	575	825	175
PlayStation	19	95	285	685	1000	210
Sony Experia	21	105	315	755	1090	440
Sony Alpha 7 III	19	95	285	685	1000	210
Prime	35	185	525	980	1400	800
Kindle	40	200	600	1120	1600	1000
Pentium Gold	26	130	390	805	1170	500
Core i7	29	155	435	900	1305	500
Xeon E	26	130	390	805	1170	500

Test Plan

These are a few of the key tests which we have decided to include in the main body of the report. You can view all of the test plans either in a web browser with google sheets here or in the separate test plan pdf document. The reason why these tests were chosen was either due to their importance to the game or to demonstrate user validation for the game. Not all the tests for each use case are shown here, but only if the tests demonstrate that the functionality works.

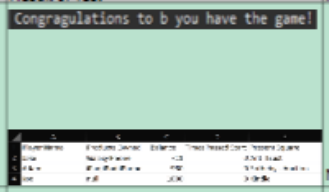
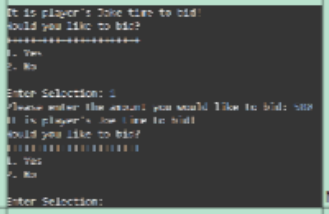
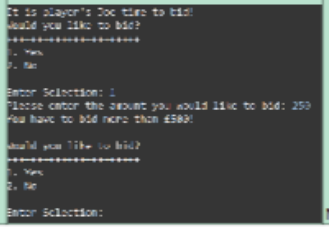
ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected	Passed?	Result of Test	Reason for failure	JUnit Test	Tester

75	Buying Product	Testing if the system offers the player the option to buy a product.	The product must not be owned by any players.	N/A	Roll the dice until the player lands on a product which is known not to be owned by another player	The player will be prompted with a message asking if they want to purchase the product	☑	`You Landed on LinkedIn`
								`This product costs`
								`Buy Product?`
								`++++++++++++`
								`1. Buy!`
								`2. Skip`
								The player is presented with the option to buy the product.
									N/A	N/A	XXXXX [X.X]

76	Buying Product	Testing if the player can purchase a product with sufficient funds.	The player must have sufficient funds and the product must not be owned by a player.	N/A	Roll the dice until the player lands on a product which is known not to be owned by another player	The player will be able to purchase the product.	☑	`Enter Selection: 0`
								`Player's name is a and their balance is = 779`
								`Owned Products:`
								`-iPad`
								`-iPod`
								`-iPhone`
								`-LinkedIn`
								The player is able to purchase the product.
									N/A	N/A	XXXXX [X.X]

77	Buying Product	Testing if the player can purchase a product without having sufficient funds.	The player must not have sufficient funds to purchase the product.	N/A	Roll the dice until the player lands on a product which is known not to be owned by another player	The player won't be able to purchase the product.	☑	`Player b's balance  = 780` The player is not able to purchase the product.	N/A	N/A	XXXXX [X.X]
78	Buying Product	Testing if a player can purchase a procuct which is owned by someone else.	The product must be owned by someone else.	N/A	Roll the dice until the player lands on a product which is known to be owned by another player	The player won't be able to purchase the product.	☑	The player is not able to purchase the product.	N/A	N/A	XXXXX [X.X]
79	Buying Product	Testing if a player can purhcase a product that they already own.	The player must already own the product.	N/A	Roll the dice until the player lands on a product which is known to be owned by the current player	The player won't be able to purchase the product again.	☑	The player is not able to purchase the product again.	N/A	N/A	XXXXX [X.X]
80	Buying Product	Testing if the system removes the correct amount from the player's balance for the purchase.	The player must land on Galaxy Tab with a balance of 1000.	N/A	Roll the dice until the player lands on a product which is known to be owned by the current player	220 will be deducted from the player's balance leaving them with a balance of 780.	☑	The correct amount is deducted from the player's balance.	N/A	N/A	XXXXX [X.X]
81	Buying Product	Testing if the system changes the players turn once they select refuse to buy the product.	The player must be given the option to buy a product.	N/A	Pass 'N' into the landedOnProduct method.	The turn will move onto the next player.	☑	The turn moves onto the next player.	N/A	N/A	XXXXX [X.X]

82	Buying Product	Testing if when a player buys a product, it adds the product to their inventory.	The first player must be given the option to buy a product.	N/A	Check the player's balance	The product will be added to the player's inventory and will be listed when the player views their inventory.	☑	`Player's name is b and their balance is = 780`
								`Owned Products:`
								`-Galaxy Tab`
								The product is added to the player's inventory.
									N/A	N/A	XXXXX [X.X]

ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected	Passed?	Result of Test	Reason for failure	JUnit Test	Tester
89	Save Game	Testing if the correct details of the players are saved into a csv file correclty	The game is started with 3 players Adam, Joe and Jake	Adam (balance = 980, present square = Anti Trust), Joe (balance = 1000, present square = Sotheby's Auction) and Jake (balance = 711, present square = Kindle). None of the players have passed the start. Adam ows ipad, ipod and iphone while Jake owns Galaxy Phone.	To save and close the game on Jake's turn	A csv file will be created with the correct inputs	☑		N/A	N/A	Adam Logan [A.L]
90	Save Game	Testing if the correct deails of the products are saved into a csv file correclty	The products Galaxy Phone, iPad, iPod and iPhone should be owned	Galxy Phone (amount to charge = 10, cost of relocation = 145), iPad (amount to charge = 9, cost of relocation = 25), iPod (amount to charge = 7, cost of relocation = 20), iPhone (amount to charge = 57, cost of relocation = 45). The products iPad, iPodm and iPhone will have 4 warehouses bought and Galaxy Phone will not have any bought. The only product to have a countyr owned is iPhone and the country should be the United States.	To save and close the game on Jake's turn	A csv file will be created with the correct inputs	☑		N/A	N/A	Adam Logan [A.L]
42	Auction starts	Testing if the current player can place their bid	Player 2 will enter their bid for Player 1s feild	500	When prompted, enter 500	Player 2 will be able to successfully enter their bid for player 1s field	☑		N/A	N/A	XXXXX [X.X]
43	Auction starts	Testing what happens when the current player, enters a lower bidding amount than the previous player	The current player will enter thier bid for the field.	250	When prompted, enter 250	The System will display an error message stating the current players bis is lower than the previous players amount, and ask them to re-enter their bid	☑		N/A	N/A	XXXXX [X.X]

ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected	Passed?	Result of Test	Reason for failure	JUnit Test	Tester
44	Auction starts	Testing to see if the system will correctly select the player with the highest bid	The system will look at all the bids and select the highest	500	System will select 500 (as it is the largest)	The system will select the correct player and this player will recieve Player 1s field	☑	It is player's Jake time to bid! Would you like to bid? ++++++++++++++++++++ 1. Yes 2. No Enter Selection: 1 Please enter the amount you would like to bid: 500 It is player's Joe time to bid! Would you like to bid? ++++++++++++++++++++ 1. Yes 2. No Enter Selection: 1 Please enter the amount you would like to bid: 250 You have to bid more than £500! Would you like to bid? ++++++++++++++++++++ 1. Yes 2. No Enter Selection: 2 You have chosen to leave the bid. The bidding is now closed. Congragulations to Jake, Ryzen is now yours!	N/A	N/A	XXXXX [X.X]
69	Player's Turn	Testing if the turn switches between four players.	It is the first player's turn.	Enter	Press 'Enter' to proceed through the game to the next player's turn.	The turn will change between the four players.	☑	The turn switches between all four players.	N/A	N/A	XXXXX [X.X]
70	Player's Turn	Testing if the player can choose other game options, i.e. purchasing a product, before ending their turn.	It is the first player's turn.	5	Pass '5' into the menu to check the player's inventory.	The player will be able to choose view their inventory before ending their turn.	☑	The player is able to choose to view their inventory before ending their turn.	N/A	N/A	XXXXX [X.X]
71	Player's Turn	Testing if the system lets eliminated players have a turn.	A player must be eliminated.	Enter	Press 'Enter' to proceed through the game to the next player's turn until it gets back to the turn of the player it started on.	The game will cycle through the turns, skipping eliminated players.	☑	The game cycles through the turns, skipping eliminated players.	N/A	N/A	XXXXX [X.X]
72	Player's Turn	Testing if the turn changes in a consistent order.	It is the first player's turn.	Enter	Press 'Enter' to proceed through the game to the next person's turn 12 times.	The turn will change in a consistent order between all players.	☑	The turn switches in a consistent order.	N/A	N/A	XXXXX [X.X]
1	Money In	Testing if a positive value is added to the user's balance and if the function will return true	The user's balance is 1000	100	Pass '100' into the method 'moneyIn(amount)'	The user's balance should be 1100 and the 'moneyIn' method should return true	☑	The amount was successfully added to the user's balance and the method returned 'True'	N/A	moneyInValidAmnt()	Adam Logan [A.L]
2	Money In	Testing if a negative value is not added to the player's balance and if the function will return false	The user's balance is 1000	-100	Pass '-100' into the method 'moneyIn(amount)'	The 'moneyIn' method should return false and the user's balance should remain unchanged	☑	The user's balance did not change and the method returned 'False'	N/A	moneyInNegativeInput()	Adam Logan [A.L]
3	Money In	Testing if nothing will be added to the user's balance if 0 is the value entered into 'moneyIn'	The user's balance is 1000	0	Pass '0' into the method 'moneyIn(amount)'	Nothing should be added to the player's balance	☑	The user's balance did not change	N/A	moneyInAddNothing()	Adam Logan [A.L]
4	Money Out	Testing if a positive value is subtracted from the user's balance and if the function will return true	The user's balance is 1000	100	Pass '100' into the method 'moneyOut(amount)'	The user's balance should be 900 and the 'moneyOut' method should return true	☑	The user's balance was 900 and the method returned 'True'	N/A	moneyOutInBalance()	Adam Logan [A.L]

ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected Results	Passed?	Result of Test	Reason for failure	JUnit Test	Tester
5	Money Out	Testing if a negative value is not subtracted from the player's balance and if the function will return false	The user's balance is 1000	-100	Pass '-100' into the method 'moneyOut (amount)'	The 'moneyOut' method should return true and the user's balance should be 1000	☐	The user's balance was 1100 and not 1000	The original check in the method simply checked if the balance was greater or equal to the amount and therefore the negative number was subtracted (which added the value to the balance as a '-, - = +') from the balance (successful test ID = 6)	moneyOutNegativeInput()	Adam Logan [A.L]
6	Money Out	Testing if a negative value is not subtracted from the player's balance and if the function will return false	The user's balance is 1000	-100	Pass '-100' into the method 'moneyOut (amount)'	The 'moneyOut' method should return false	☑	The user's balance was 1000	N/A	moneyOutNegativeInput()	Adam Logan [A.L]
7	Money Out	Testing if nothing will be subtracted to the user's balance if 0 is the value entered into 'moneyOut'	The user's balance is 1000	0	Pass '0' into the method 'moneyOut (amount)'	Nothing should be subtracted from the player's balance	☑	The user's balance did not change	N/A	moneyOutRemoveNothing()	Adam Logan [A.L]
8	Money Out	Testing if the user can lose all the money in their balance and if the function will return true	The user's balance is 1000	1000	Pass '1000' into the method 'moneyOut (amount)'	The user's balance should be 0 and the 'moneyOut' method should return true	☑	The user's balance was 0 and the method returned 'True'	N/A	moneyOutRemoveMax()	Adam Logan [A.L]
9	Money Out	Testing if the 'moneyOut' function will return false if the user loses more money than there is in their account	The user's balance is 1000	1001	Pass '1001' into the method 'moneyOut (amount)'	The user's balance should be -1 and the 'moneyOut' method should return false	☑	The user's balance is -1 and the method returned 'False'	N/A	moneyOutRemoveOverBalance()	Adam Logan [A.L]
31	Lost Stock Exchange	Testing if the correct amount is returned when the user incorrecity guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	1	pass 1 as the guessedNum, 100 as the invtAmnt and 5 as the correctGuess into the 'playGame()' method	The returned value should be -40	☑	Returned -40	N/A	loseBy4()	Adam Logan [A.L]
32	Lost Stock Exchange	Testing if the correct amount is returned when the user incorrecity guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	2	pass 2 as the guessedNum, 100 as the invtAmnt and 5 as the correctGuess into the 'playGame()' method	The returned value should be -30	☑	Returned -30	N/A	loseBy3()	Adam Logan [A.L]
33	Lost Stock Exchange	Testing if the correct amount is returned when the user incorrecity guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	3	pass 3 as the guessedNum, 100 as the invtAmnt and 5 as the correctGuess into the 'playGame()' method	The returned value should be -20	☑	Returned -20	N/A	loseBy2()	Adam Logan [A.L]
34	Lost Stock Exchange	Testing if the correct amount is returned when the user incorrecity guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	4	pass 4 as the guessedNum, 100 as the invtAmnt and 5 as the correctGuess into the 'playGame()' method	The returned value should be -10	☑	Returned -10	N/A	loseBy1()	Adam Logan [A.L]

ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected	Passed?	Result of Test	Reason for failure	JUnit Test	Tester
35	Lost Stock Exchange	Testing if the correct amount is returned when the user incorreclty guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	6	pass 6 as the guessedNum, 100 as the invtAmnt and 1 as the correctGuess into the 'playGame()' method	The returned value should be -50	☑	Returned -50	N/A	loseBy5()	Adam Logan [A.L]
36	Lost Stock Exchange	Testing if the correct amount is returned when the user incorreclty guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	7	pass 7 as the guessedNum, 100 as the invtAmnt and 1 as the correctGuess into the 'playGame()' method	The returned value should be -60	☑	Returned -60	N/A	loseBy6()	Adam Logan [A.L]
37	Lost Stock Exchange	Testing if the correct amount is returned when the user incorreclty guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	8	pass 8 as the guessedNum, 100 as the invtAmnt and 1 as the correctGuess into the 'playGame()' method	The returned value should be -70	☑	Returned -70	N/A	loseBy7()	Adam Logan [A.L]
38	Lost Stock Exchange	Testing if the correct amount is returned when the user incorreclty guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	9	pass 9 as the guessedNum, 100 as the invtAmnt and 1 as the correctGuess into the 'playGame()' method	The returned value should be -80	☑	Returned -80	N/A	loseBy8()	Adam Logan [A.L]
39	Lost Stock Exchange	Testing if the correct amount is returned when the user incorreclty guesses the number	The stock exchange object has been set with the range 1 - 10 and a risk of 0.25. The correct guess is 5 and the investement is 100.	10	pass 10 as the guessedNum, 100 as the invtAmnt and 1 as the correctGuess into the 'playGame()' method	The returned value should be -90	☑	Returned -90	N/A	loseBy9()	Adam Logan [A.L]
83	Eliminate Player	Testing if the correct player is eliminated.	One player must be called "a".	a	Pass "a" into the eliminatePlayers method.	The specified player will be eliminated.	☑	The correct player is eliminated.	N/A	N/A	XXXXX [X.X]
84	Eliminate Player	Testing if the player is eliminated once their balance reaches 0.	Player's balance must be at 5 when they land on FitBit.	Y	Pass 'Y' into the landedOnProduct method.	The player will be eliminated after they purchase the product.	☑	The player is eliminated after they purchase the product.	N/A	N/A	XXXXX [X.X]
85	Eliminate Player	Testing if the products owned by the player are returned to the system once the player is eliminated.	Player must own at least one product when they are eliminated	2	Pass '2' into the menu to eliminate the player.	The products owned by the player will be returned to the system.	☑	The products owned by the player are returned to the system.	N/A	N/A	XXXXX [X.X]
86	Winning	Testing if the win message displays correctly when a player wins.	There must be two players left.	2	Pass '2' into the menu to eliminate a player.	The win message will display correctly.	☑	`Congragulations to b you have the game!` The win message displays correctly.	N/A	N/A	XXXXX [X.X]
87	Winning	Testing if a player wins once they are the last player left.	There must be two players left.	2	Pass '2' into the menu to eliminate a player.	The player will win when they are the last one left.	☑	`Congragulations to b you have the game!` The player wins when they're the last one left alive.	N/A	N/A	XXXXX [X.X]
88	Winning	Testing if the game ends once a player wins.	A player must have won.	Enter	When the win message is displayed, input 'Enter'.	The game will end and the application will close.	☑	`Congragulations to b you have the game!` The game ends when a player wins.	N/A	N/A	XXXXX [X.X]

ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected	Passed?	Result of Test	Reason for failure	JUnit Test	Tester
54	Request to buy country	Testing to see if the player can see the option to request to purchase a country, if they dont have all 4 warehouses on the products	The player will select the 'request to buy country' option and will own all then products within the company 'AMD', but doesnt own all 4 warehouses on all the products	Selects to buy	The player wont be given the option to purchase a country when they dont own all the warehouses on the products	The option to request a country will not be shown as the player does not own all the warehouses, on their products, within 'AMD'	☑	It is Jake's go! Player Options ++++++++++++++ 1. Roll dice 2. Check inventory 3. Help 4. Leave Game 5. Close Game 6. Save and Close Game 7. Purchase a warehouse Enter Selection: 2 Your name is Jake and your balance is = £1000 Owned Products: -SeaMicro with 4 warehouse(s) -Ryzen with 0 warehouse(s)	N/A	N/A	XXXXX [X.X]
55	Request to buy country	Testing to see if, when the player tries to purchase another country, the same option will not appear twice	The player will select 'request to purchase a country' and wont be given the option 'United States', as the player already has a product with this county	Selects to buy	the player will select the 'request to buy office' option. The system will check to see if the player already owns any other countries on other products.	Since the player already owns 'United States' on another product, the system wont display this option again to the user, making them choose a different option. the relocation fee has also increased by £25	☑	Which product would you like to relocate? ++ 1. SeaMicro (Cost = 70) 2. Ryzen (Cost = 50) 3. Cancel Enter Selection: 1 Which country would you like to relocate too? ++ 1. United Kingdom 2. Canada 3. France 4. Germany 5. Ireland 6. Italy 7. Japan Enter Selection:	N/A	N/A	XXXXX [X.X]
56	Charged Product	Testing to see if player is charged money when they land on owned product	Both users balance start with 1000	1 (Roll dice)	Enter 1 (Land on an owned product)	The player landing on the owned product will have their balance reduced, the player owning product will receive their balance	☑	You have landed on a product someone else owns! You have been charged 9 to use iPad. Player one's balance = 1009 Player two's balance = 991	N/A	N/A	XXXXX [X.X]
57	Charged Product	Testing if player will be charged more if product has warehouse	Player with product must also have a warehouse	1 (Roll dice)	Enter 1 (Land on an owned product)	Since player 1 has added a warehouse to their product, it should charge the next player who lands on it	☑	You have landed on a product someone else owns! You have been charged 45 to use iPad.	N/A	N/A	XXXXX [X.X]
58	Charged Product	Testing if product with a country will charge more	Player with product must own a country	1 (Roll dice)	Enter 1 (Land on an owned product)	When player that owns a product upgrades and gets a country, it should charge the other players landing on it	☑	You Landed on iPad You have landed on a product someone else owns! You have been charged 79 to use iPad.	N/A	N/A	XXXXX [X.X]

ID	Use Case	Description of Test	Test Initialisation	Test Inputs	Test Procedure	Expected	Passed?	Result of Test	Reason for failure	JUnit Test	Tester
60	Request Warehouse	When choosing a product to purchase a warehouse for enter string instead of int	Player must own a product	f	Choose option 'Purchase Warehouse' then enter "f" for answer	The system should reject the input and return an error message and allow them to re enter a number.	☑	Choose a Product to dedicate a warehouse too ++ 1. iPad (Cost = £22) 2. iPod (Cost = £17) 3. iPhone (Cost = £17) 4. Cancel Enter Selection: f Error input. Enter Selection:	N/A	N/A	XXXXX [X.X]
61	Request Warehouse	Continue entering incorrect values when purchasing warehouse	Must be purchasing a warehouse	5	Choose the option to purchase the warehouse enter integer out of range, letter and larger integers.	The system should display an error message	☑	Choose a Product to dedicate a warehouse too ++ 1. iPad (Cost = £22) 2. iPod (Cost = £17) 3. iPhone (Cost = £17) 4. Cancel Enter Selection: 5 Enter a value between 1 and 4 Enter Selection:	N/A	N/A	XXXXX [X.X]
62	Request Warehouse	Continue entering incorrect values when purchasing warehouse	Must be purchasing a warehouse	55	Choose the option to purchase the warehouse enter integer out of range, letter and larger integers.	The system should display an error message	☑	Choose a Product to dedicate a warehouse too ++ 1. iPad (Cost = £22) 2. iPod (Cost = £17) 3. iPhone (Cost = £17) 4. Cancel Enter Selection: 55 Enter a value between 1 and 4 Enter Selection:	N/A	N/A	XXXXX [X.X]
63	Request Warehouse	Continue entering incorrect values when purchasing warehouse	Must be purchasing a warehouse	1000000	Choose the option to purchase the warehouse enter integer out of range, letter and larger integers.	The system should display an error message	☑	Choose a Product to dedicate a warehouse too ++ 1. iPad (Cost = £22) 2. iPod (Cost = £17) 3. iPhone (Cost = £17) 4. Cancel Enter Selection: 1000000 Enter a value between 1 and 4 Enter Selection:	N/A	N/A	XXXXX [X.X]
64	Request Warehouse	Continue entering incorrect values when purchasing warehouse	Must be purchasing a warehouse	1000000000000000000000000000000000000000000000000	Choose the option to purchase the warehouse enter integer out of range, letter and larger integers.	The system should display an error message	☑	Choose a Product to dedicate a warehouse too ++ 1. iPad (Cost = £22) 2. iPod (Cost = £17) 3. iPhone (Cost = £17) 4. Cancel Enter Selection: 1000000000000000000000000000000000000000000000000 Error input. Enter Selection:	N/A	N/A	XXXXX [X.X]

Appendix

Weekly Team Minutes [A.L]

24/1/2022

Minutes for CSC2058 Project 45 Week commencing 24/1/2022 Date of this minute 28/1/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X.H
XXXXX	X.X

Task Reporting

N/A as this was the first time we meet after the submission of the first deliverable.

Interim Deliverables:

N/A as this was the first time we meet after the submission of the first deliverable.

Actions Taken During Meeting

During this meeting the team discussed what had to be implemented and divided these tasks amongst ourselves.

Actions Planned

The following features will need to be implemented:
Adam Logan:

- Stock Exchange
- Build Warehouse

XXXXX:

- Check Inventory
- Eliminate Player when they have no money

XXXXX:

- Request Country
- Product Bidding

XXXXX:

- Income Event
- Make Menu Only Display Valid Options

31/1/2022

Minutes for CSC2058 Project 45 Week commencing 31/1/2022 Date of this minute 4/2/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X
XXXXX	X.X

Task Reporting

The following functions features have been implemented by the respective individual below:
Adam Logan:

- Stock Exchange
- Build Warehouse

XXXXX:

- Check Inventory

XXXXX:

- Request Country

Interim Deliverables:

Adam Logan:
https://gitlab2.eeecs.qub.ac.uk/CSC2058-2122/csc2058-2122-g45/-/commit/28661d44cb431eaf2448caebf6f2ff1d2769b6cf

XXXXX & XXXXX:
Did not push the code to the repository and therefore cannot be shown.

Actions Taken During Meeting

During this meeting we discovered that several of the team members had technical issues with their chosen editors and with using git. This therefore meant that they were unable to either complete the code or to push their code to the gitlab repository. There were also several impediments raised regarding the implementation of a few features.

Actions Planned

All team members will meet again to make sure that git is working on everyone's machine. During the next meeting all current impediments will be sorted together as a team.

Adam Logan:

- To merge the code completed by XXXXX and XXXXX into the gitlab repository (specifically into the master branch as no other branch exists at this time)

XXXXX, XXXXX and XXXXX:

- No action needs to be taken

7/2/2022

Minutes for CSC2058 Project 45 Week commencing 07/2/2022 Date of this minute 08/02/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X

Task Reporting

Adam Logan:

- Merged XXXXX's code

Interim Deliverables:

Adam Logan:
https://gitlab2.eeecs.qub.ac.uk/CSC2058-2122/csc2058-2122-g45/-/commit/3fbcafb17dcc10d25f6843bec5e78a5f8964df4b

Actions Taken During Meeting

During this meeting the team worked on a way to check if a player owns all the products of the company and can be seen here. Adam Logan demonstrated to the other team members how to use git bash, which XXXXX tested on his machine. The remainder of the meeting was attempting to fix XXXXX's eclipse editor to allow it to run the code which was pulled from the repository.

Actions Planned

XXXXX:

- To work on the auction aspect of the 'product bidding' game

XXXXX:

- To implement the functionality for the player to request a country

14/2/2022

Minutes for CSC2058 Project 45 Week commencing 14/2/2022 Date of this minute 18/2/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X.H
XXXXX	X.X

Task Reporting

XXXXX:

- Started Working on the product bidding game

Interim Deliverables:

XXXXX:
https://gitlab2.eeecs.qub.ac.uk/CSC2058-2122/csc2058-2122-g45/-/commit/043e7f2a4b1a9da153df5029f4120a8a9b5e3426

Actions Taken During Meeting

Discussed the general progress of the project. The team also attempted to assist XXXXX in allowing his eclipse editor to run the code pulled from the gitlab repository. This was unsuccessful so it was decided that XXXXX would work on the 'description of the text user-interface' section of the project. It was also decided that Adam Logan and XXXXX would meet together to complete the product bidding functionality, as this was deemed to difficult for one member to work on alone. As time was running short it was also decided that the team would meet again on the 21/1/2022 to discuss the test plan.

Actions Planned

Adam Logan and XXXXX:

- To work on the product bidding aspect of the game together at a later date

XXXXX:

- To work on the description of the text user-interface

XXXXX:

- To continue working on the income Event
- To continue working on the making menu only display valid options

21/2/2022

Minutes for CSC2058 Project 45 Week commencing 21/2/2022 Date of this minute 21/2/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X

Task Reporting

N/A

Interim Deliverables:

N/A

Actions Taken During Meeting

During this meeting, the two team members present discussed the test plan and what format these tests descriptions should take. What was also discussed was how JUnit tests would be incorporated into the project and the general rules which we would abide by when testing.

Actions Planned

Adam Logan and XXXXX:

- To start working on test descriptions within the test plan

28/02/2022

Minutes for CSC2058 Project 45 Week commencing 28/2/2022 Date of this minute 4/3/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X

Task Reporting

N/A

Interim Deliverables:

N/A

Actions Taken During Meeting

This was a pair programming session with the two team members present, to complete the functionality of the product bidding game. It was decided that XXXXX would be the driver and Adam Logan would be the instructor as Adam Logan is the more experienced programmer and therefore would allow XXXXX to learn more about the code base.

Actions Planned

N/A

7/3/2022

Minutes for CSC2058 Project 45 Week commencing 7/3/2022 Date of this minute 7/3/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X

Task Reporting

Adam Logan:

- Completed the test plan for tests 1 – 36
- Completed testing for tests 1 - 23

XXXXX:

- Completed the test plan for tests 37 - 45

Actions Taken During Meeting

During this meeting the team divided up the use cases which each member of the team would complete the test plan for and eventually test.

Actions Planned

Adam Logan:

- To complete the sequence diagrams for this deliverable

XXXXX:

- To work on test plans for the use cases:
    - Charged Product
    - Request to buy warehouse
    - System builds warehouse

XXXXX:

- To work on the text user interface descriptions
- Continue working on test plans for the use cases:
    - Auction Starts
    - Request to buy country
    - Player Buys Country
    - Create Player
    - Income Event

XXXXX:

- To work on test plans for the use case:
    - Players Turn
    - Passing Go
    - Buy Product
    - Player Eliminated
    - Winning

Minutes for CSC2058 Project 45 Week commencing 7/3/2022 Date of this minute 11/3/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X

Task Reporting

Adam Logan:

- Completed the sequence diagrams for this deliverable

XXXXX:

- Completed test plans for the use cases:
  - Charged Product
  - Request to buy warehouse
  - System builds warehouse

XXXXX:

- To work on the text user interface descriptions
- Completed test plans for the use cases:
  - Auction Starts
  - Request to buy country
  - Player Buys Country
  - Create Player
  - Income Event

XXXXX:

- Completed test plans for the use cases:
  - Players Turn
  - Passing Go
  - Buy Product
  - Player Eliminated
  - Winning

Interim Deliverables:

Adam Logan:

https://drive.google.com/drive/folders/1ezdzXw5EuQQx81pA7BcRHCo4CtTkxMzL?usp=sharing

Actions Taken During Meeting

During this meeting the team reviewed the test plans each member completed for the meeting.

Actions Planned

Adam Logan:

- To update the use cases and the use case diagram

XXXXX:

- To work on the text user interface descriptions

Adam Logan, XXXXX, XXXXX and XXXXX:

- To test the use cases which each member completed the test plans for

14/3/2022

Minutes for CSC2058 Project 45 Week commencing 14/3/2022 Date of this minute 19/3/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X

Task Reporting

Adam Logan:

- To update the use cases and the use case diagrams

XXXXX & XXXXX & XXXXX:

- Completed tests

Interim Deliverables:

Adam Logan:
https://docs.google.com/document/d/1uuh3AgFo5pGVennUAqzn2zZBW_L_5fQoxLj8Txs2NyU/edit?usp=sharing

Actions Taken During Meeting

During this meeting the team discussed what is left to do for the project and reviewed the tests that were complete for the meeting. We also discussed what was needed to be included within the demo video.

Actions Planned

Adam Logan:

- Complete security considerations and development process

XXXXX:

- Complete the demo video

XXXXX:

- Complete the text user interface

XXXXX:

- Review code output for spelling mistakes and to write the instructions/rules of the game

Minutes for CSC2058 Project 45 Week commencing 21/3/2022 Date of this minute 25/3/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X

Task Reporting

N/A

Interim Deliverables:

N/A

Actions Taken During Meeting

During this pair programming session, we took some time to complete the request country method. It was decided that Adam Logan would be the driver and that XXXXX would be the instructer for this. This was mainly due to the fact that XXXXXs programming software was not working correctly and so Adam Logan was made the Driver.

Actions Planned

N/A

21/3/2022

Minutes for CSC2058 Project 45 Week commencing 21/3/2022 Date of this minute 25/3/2022

The following team members were present on Teams when minutes were discussed:

Name	Signature
Adam Logan	A.L
XXXXX	X.X
XXXXX	X.X

Task Reporting

Adam Logan:

- Completed security Consideration and finished development process document

XXXXX:

- Finished recording clips and editing of the Interim video

XXXXX:

- Finished text user interface

XXXXX:

- Added to the development process document
- Reviewed code for any grammar mistakes and corrected it

Interim Deliverables:

Adam Logan & XXXXX
https://docs.google.com/document/d/1vYDWDGqqrf8z-Ji7IVZO5CDklhyIuoG9lGR4jr4SKN4/edit
XXXXX:
https://docs.google.com/document/d/1fskcSGJXEbxTqfezs4P5inGBLSQRIjT9/edit?usp=sharing&ouid=103168906336398527104&rtpof=true&sd=true

Actions Taken During Meeting

We suggested meeting up several times to ensure all work is completed before deadline.
During these meetings we disccuded who was doing what part of the documentation. Decided if any part of the program needed to be adjusted for any errors of mistakes. After proof reading everything we then planned to upload the assignement.

Actions Planned

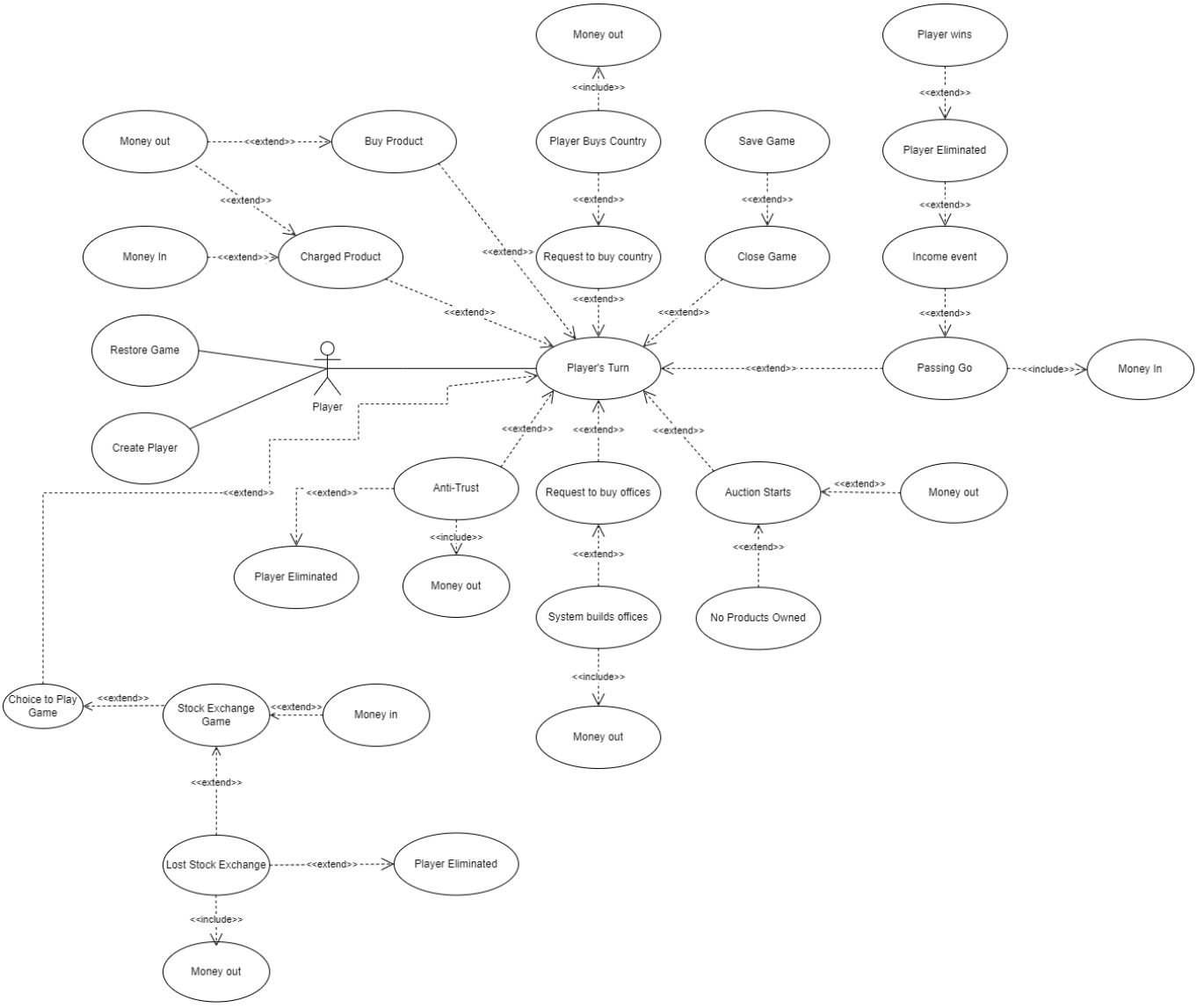
Team:

- N/A

No additional plans were made as this was the final week and the assignment was finished.

Updated Use Cases [A.L]

Use Case Diagram [A.L]



Updated Use Case Descriptions

Stock Exchange Game [A.L]	
Objective	The player will guess a number between
Precondition	A player has chosen to play the Stock Exchange game
Main Flow	1. A random number is generated between the range specified 2. A message is displayed to the player prompting them to select a number between the range 3. The player enters a number

Alternative Flows	• At 3 the player:
	  ○ Selects the correct number and the 'Money In' use case is triggered
	  ○ Selects an incorrect number and the 'Lost Stock Exchange' use case is triggered

Post-condition	The player will either have earned money or have lost money

Close Game [A.L]	
Objective	To end the game
Precondition	A player has chosen to end the game prematurely
Main Flow	1. System displays a generic goodbye message 2. The system ends the game

Alternative Flows	• At 1:
	  ○ If the player chooses to save the game the 'Save Game' use case is triggered
	  ○ If the player did not choose to save the game the player(s) with the highest balance are displayed as the winners

Post-condition	The game is closed

Save Game [A.L]	
Objective	The state of the current game will save
Precondition	A player has chosen save the game
Main Flow	1. System gets all the relevant information from the game 2. System saves this information in a csv file
Alternative Flows	N/A
Post-condition	The state of the game will have been saved

Restore Game [A.L]	
Objective	The saved game is loaded into the current game
Precondition	A player has chosen to save the game
Main Flow	1. System retrieves all the stored data 2. System updates the objects of this system with this data
Alternative Flows	N/A
Post-condition	The game saved on the csv is now playable

Player's Turn [A.L]	
Objective	The player will be given options and then they will roll the die
Precondition	It is the player's turn
Main Flow	1. The player will be given the options to leave the game, to close the game entirely or to check their inventory 2. The system will 'roll the dice' by randomly generating two random numbers between 1 and 6 3. Player will be moved to the area counting the total from their current position
Alternative Flows	• At 1 the player chooses to purchase: ○ If the player owns a product, they will be given an option to purchase a warehouse, and if they choose this option the 'Request to buy warehouses' use case is triggered ○ If the player owns all the products in a company, they will be given an option to purchase a warehouse, and if they choose this option the 'Request to buy warehouses' use case is triggered ○ If they choose to leave the game the player will be eliminated ○ If they choose to close the game the game will end for all players ○ If they choose to check their inventory the player's balance and owned products will be displayed • At 2: ○ If the system generates two of the same number this step repeats itself ○ If the system generates two of the same number three times in a row, then the player is moved to the auction square • At 3 if the player lands on the: ○ Product area the 'Lands on product' use case is triggered'
Post-condition	Player rolls dice and two numbers are randomly generated

Auction Starts [A.L]	
Objective	One of the player's products goes up for auction for other users to bid on.
Precondition	Player lands on specific square on board
Main Flow	1. System puts one of the player's products up for auction 2. System takes turns to ask users how much they would like to bid. 3. At the end of the auction, once 1 player is left, System then determines who had the highest bid. 4. System transfers ownership of the product to the winning player and charges the winning player the amount of the bid
Alternative Flows	• At 1: ○ If the player doesn't have a company to auction up, then a message is displayed
Post-condition	Company will be auctioned, and winning player will get the company

Game Description [X.X, X.X]

The game begins with all players on the red ‘Start of Quarter’ square on the bottom right of the board, the same place as ‘GO’ in monopoly. Players take turns to roll the dice and move accordingly with the number that they roll. A player can land on two different types of squares, one being products the other being events. When a player lands on a product square which hasn’t been purchased already, they will be given an option to buy this product.

One of the event squares is ‘Product Bidding’, when a player lands on this their products are put up for auction, meaning that all players on the board can bid on a product of the player who landed on the square. Whichever player bids the highest will receive ownership of the product.

When a player lands on a square which is owned by another player, they are charged for the use of the product, with the amount charged being dependent on what the product is. Some products will not charge players too much to use, while some will be more expensive. When a player wants to build a warehouse, provided they can, they are charged for each warehouse they build. When a player lands on a square with a warehouse on it, they are charged an increased rate on what the standard cost to use the product would be, this gives an incentive for players to build warehouses. The reason why there is an increased rate is due to the product being improved by them being worked on in the warehouse.

When a player has built warehouses on all squares of a field, they are then given the option to select a region. Selecting a region comes at a cost, but it also comes with a reward as it allows a player to expand their business to the whole region, meaning that when a player lands on their square, they will have to pay an even higher rate than before. There are different regions that are available to select, and each has its own pros and cons, for example some regions will cost more to build on but will charge more money for players who land on their squares.

Each time a player passes the ‘Start of Quarter’ square they receive a bonus. When a player passes the ‘Start of Quarter’ square four times, marking that a full year has passed, the income event will trigger. This event checks the player’s income for the year, and if it is below a certain amount then the player is eliminated from the game. The amount of income required will depend on how many players are left in the game, and when a player is eliminated, the amount will go up as the remaining players will have had the opportunity to purchase the products which were owned by the eliminated player.

There are ‘Stock Exchange’ event squares around the board, and when a player lands on them they can decide to take a gamble, picking a number between a certain range; if the player guesses the same number as the computer then they will double their money, but if they’re wrong then they will lose money, the amount they lose depends on how far away their guess was from the value produced by the computer. When a player decides on how much they are going to contribute to investments, they can sometimes lose more money than they put in

There is a ‘Tax’ event square which when landed on, charges the player 10% of their current value. There is also a ‘Bank Holiday’ square which does nothing, this is not very useful in the beginning of the game, but towards the endgame it means the player gets a free pass on their turn instead of landing on a product owned by someone else and having to pay them.

Each product has its own unique value, and this value changes with each upgrade that is purchased. The last player left on the board will be the winner after all other players have been eliminated.

Whenever a player is stuck with how to play the game, they can request to see instructions during their turn in the menu if they missed out on seeing it at the start. If the players wish to do so they are also able to save the state of the current board and player inventories, allowing them to pick up where they left off when they closed the program.

Example Play Through

You can view an example play through of the game here.

Security Considerations [A.L, X.X]

An improvement to our project that could have been made, regarding security, is encrypting the details of the saved game. This would improve the integrity of the game allowing the users to be confident that no changes have been made. This would also ensure the confidentiality of the names of the users that are saved within the file. A checksum could also be added to the file once again ensuring integrity as if the file has been changed the system will be able to detect this.

If encryption was used this would also allow multiple games to be saved, being password protected. These passwords would be hashed to ensure their security. A vulnerability which could be exploited by this is if an attacker uses a rainbow table. This threat could be mitigated by using a large random salt value for each password.

If the game were to be online and the ability to save games were to be kept then, for the login process, measures would need to be taken to prevent an SQL injection attack from taking place. This would need to take place as it is most likely that this data would then need to be stored within a database. To prevent an SQL attack, we would create a stored procedure for login instead of entering the user input as SQL code.

An additional feature that could be added to the program would be auditing. Incorporating audits for the application would allow users with higher access to examine events that happened during the use of the program. The audit would show actions such as saving, loading, and changing data. These audits can be used as security and can also help with data collection when debugging for future updates.

In case of a catastrophic failure for instance the computer crashing, a network connection error or a disk drive fault. Future development of the program could include an autosave function. This would mean all data will be successfully stored in a separate file, even if a computer closes the program without the user manually saving the state of the game. This ensures the integrity of the program.

Development Process [A.L, X.X]

As mentioned within the team minutes there were several instances of pair programming that took place throughout the process of implementation. The reason for this was to allow the less experienced programmers to get accustomed to the code base, so all members of the project would be familiar, and comfortable, with the code.

Another reason why pair programming took place was due to one of the team members experiencing technical difficulties and not being able to run the code. Instead of this team member not being able to contribute to the code, it was decided that pair programming was the ideal solution to this problem.

To review the code each member contributed we decided to do mob programming in an attempt to find small errors, such as spelling mistakes, and to suggest changes to the code which the original contributor may not have noticed or thought of. This was to provide the optimal solution to the problem at hand.

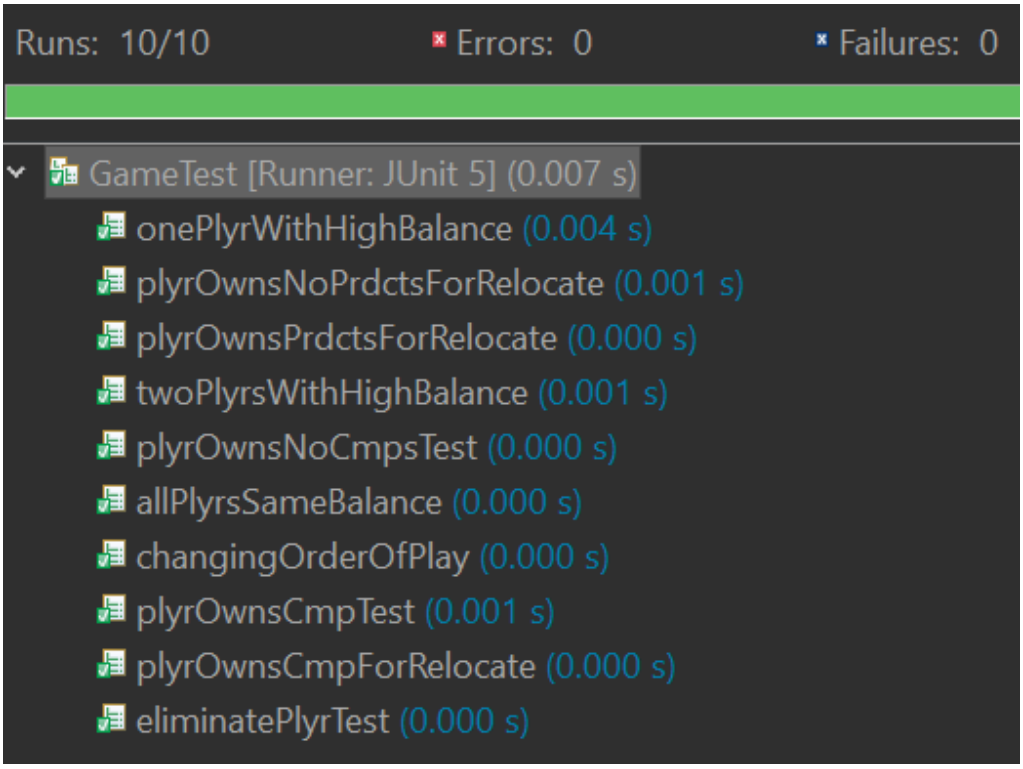
The development process we followed is called Rapid Application Development (RAD), the reason we chose to follow this development process is because it allowed us to be flexible with our schedules and it made it very easy to divide up the workload between the four members of the group. As we were following RAD, each member was able to work on their own individual methods and sections of the code, testing as they were going along to make sure each part of the code was working as more was being added. This development process worked very well for our group as the fast-paced nature of the process meant it was very easy to get the program written and tested with all four of us working on it. This approach was best suited to our team as it allowed us to test our code as we were writing it, which we would not have been able to do with popular approaches such as the waterfall development process as it only allows for testing after all the code has been written and can only be retested after all the issues have been addressed, making it too slow for it to work for our group. Rapid application development also helped in the sense that nobody had to wait on anyone else to finish their code before testing as each member was able to test their code as they were writing it, this is a much faster way of doing it and made it very workable for our members' busy schedules. Another advantage of following this development process is that it allowed us to include a few extra features which weren't technically needed but added to the experience of the game, a good example of this is the save game feature; this was possible because of the fast-paced development promoted by the rapid application development process.

We were forced to deviate from the rapid application development process towards the end as after we began the construction of the program, it became clear that some methods needed adjusting so that they could work properly and efficiently together, so we did have to go back on ourselves once but that was the only time through the whole semester that our team had any issues with following this development process. If our group was tasked to do this or something similar again, we have all agreed that we would follow the rapid application development process again because of how well it worked for us in terms of managing the workload and just the overall pace of development.

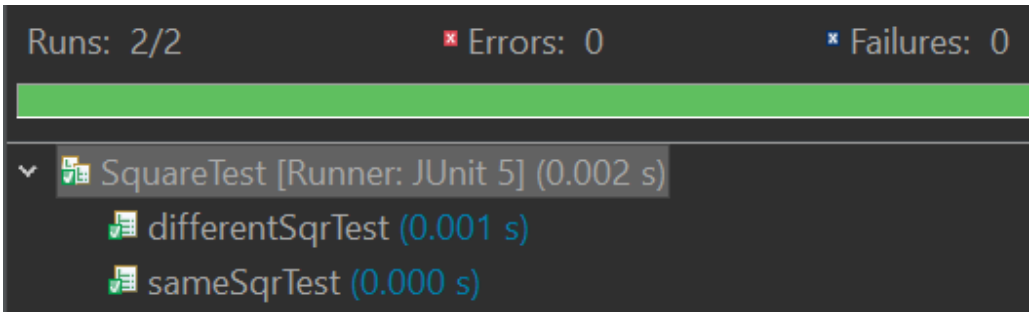
Testing

Junit Test Results

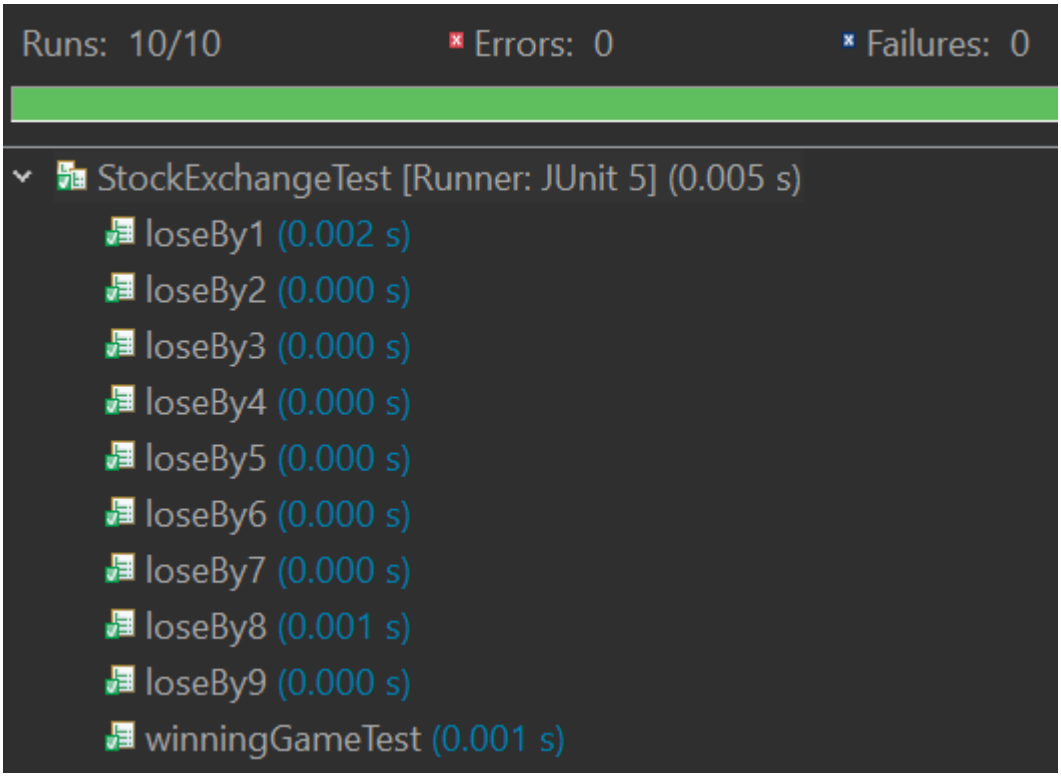
Game Test



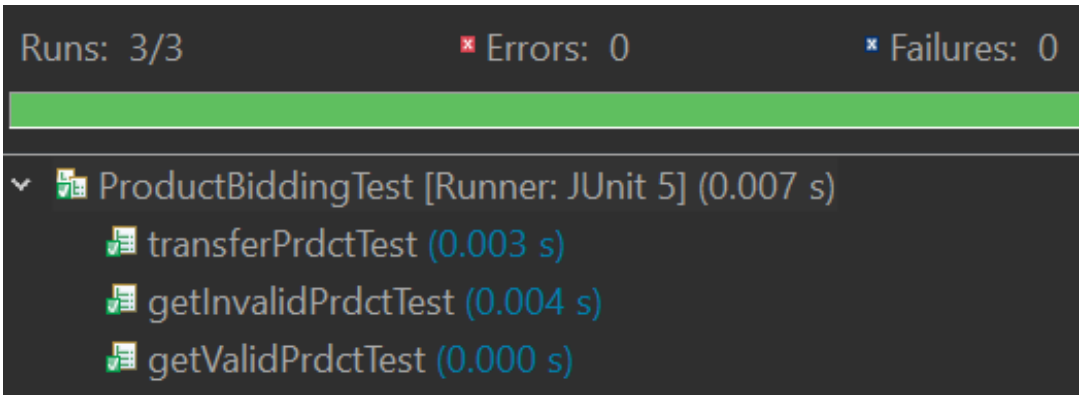
Square Test



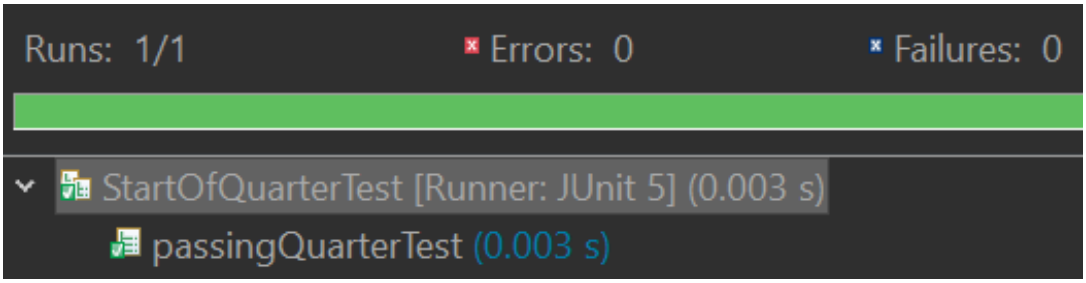
StockExchange Test



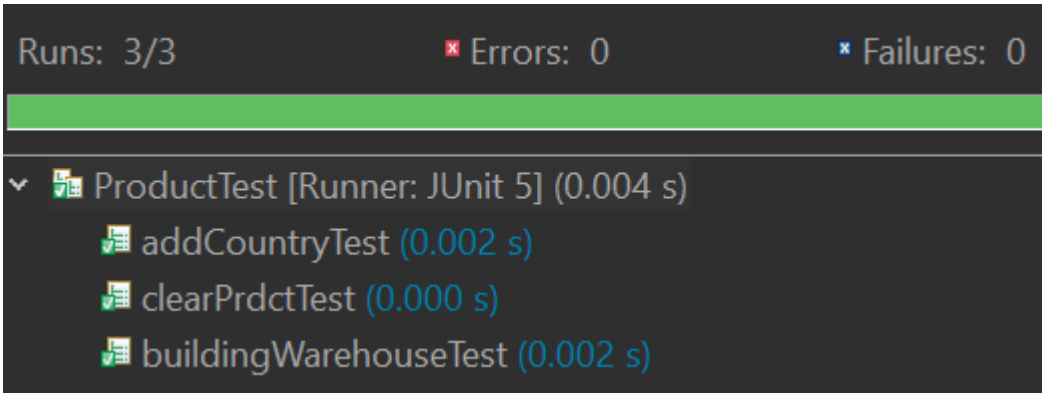
ProductBidding Test



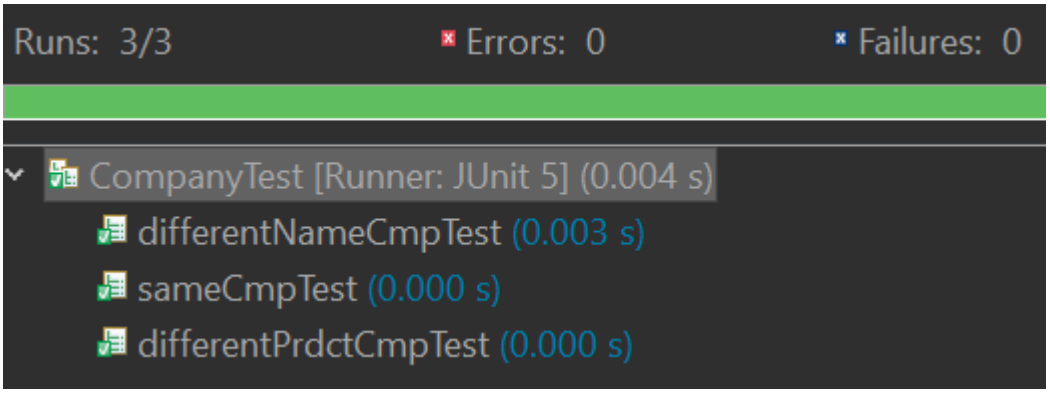
StartOfQuarter Test



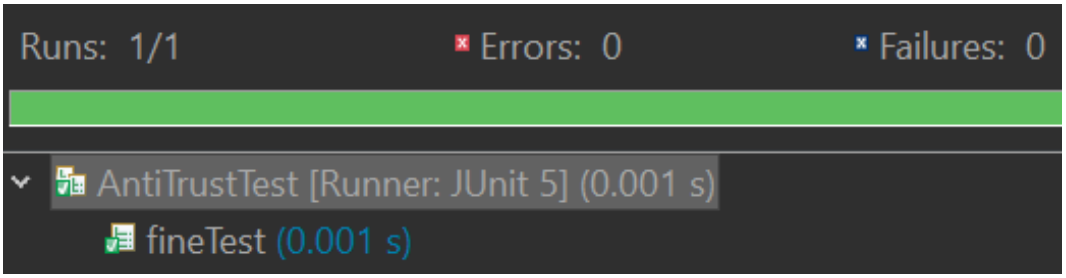
Product Test



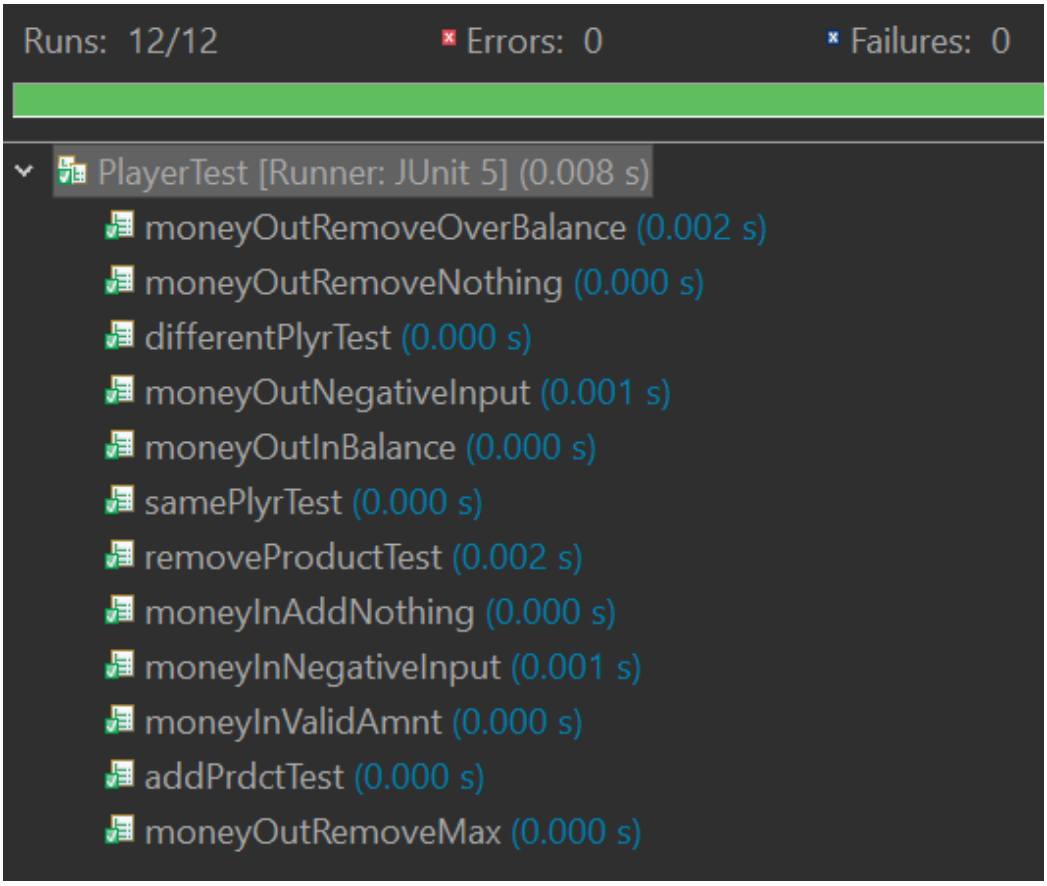
Company Test



AntiTrust Test



Player Test


Runs: 12/12
Errors: 0
Failures: 0
PlayerTest [Runner: JUnit 5] (0.008 s)
moneyOutRemoveOverBalance (0.002 s)
moneyOutRemoveNothing (0.000 s)
differentPlyrTest (0.000 s)
moneyOutNegativeInput (0.001 s)
moneyOutInBalance (0.000 s)
samePlyrTest (0.000 s)
removeProductTest (0.002 s)
moneyInAddNothing (0.000 s)
moneyInNegativeInput (0.001 s)
moneyInValidAmnt (0.000 s)
addPrdctTest (0.000 s)
moneyOutRemoveMax (0.000 s)

Test Coverage

Game Test Coverage

GameTest.java

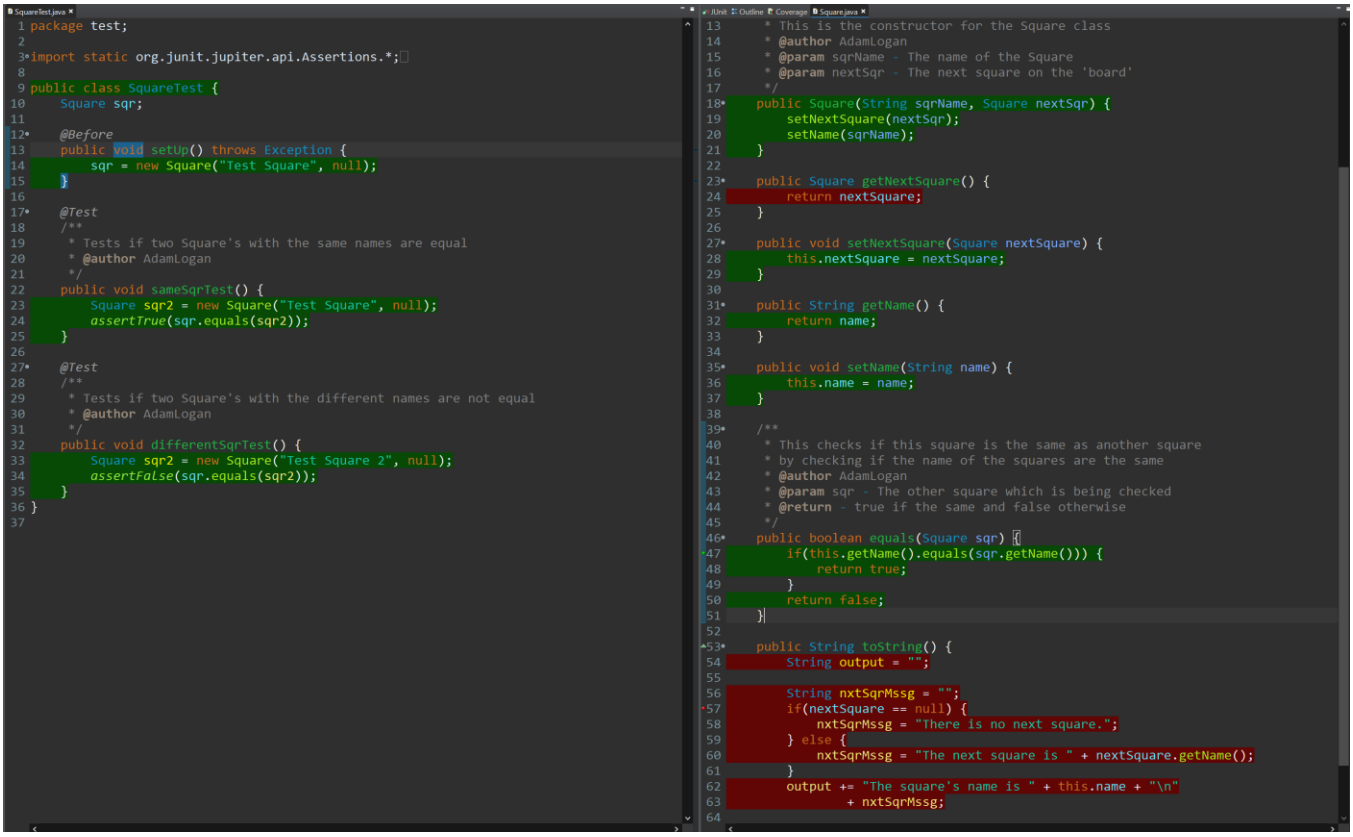
```java
package test;

import static org.junit.jupiter.api.Assertions.*;

public class GameTest {
    Player plyr1 = new Player("Test player 1");
    Player plyr2 = new Player("Test player 2");
    Player plyr3 = new Player("Test player 3");
    Player[] allPlayers = {plyr1, plyr2, plyr3};

    Company cmp;
    Company[] allCmps = new Company[1];

    @Before
    public void setUp() throws Exception {
        Product prdct1 = new Product("Test Product 1", null, 100, 100, new int[] {1,2,
        Product prdct2 = new Product("Test Product 2", null, 100, 100, new int[] {1,2,
        Product[] allPrdcts = {prdct1, prdct2};

        cmp = new Company("Test Company", allPrdcts);

        allCmps[0] = cmp;

        plyr1.setOwnedProducts(allPrdcts);
    }

    @Test
    /**
     * Tests if the order of the array has been changed correctly
     * @author AdamLogan
     */
    public void changingOrderOfPlay() {
        Player[] newOrder = {plyr2, plyr3, plyr1};

        allPlayers = Game.orderOfPlay(allPlayers, 1);

        assertTrue(arryEqual(newOrder, allPlayers));
    }

    @Test
    /**
     * Tests if the two Players with the highest balance are returned
     * @author AdamLogan
     */
    public void twoPlyrsWithHighBalance() {
        plyr1.setBalance(1001);
        plyr2.setBalance(1001);

        ArrayList<Player> highestPlayers = Game.highestBalance(allPlayers);

        ArrayList<Player> crrtHighestPlyrs = new ArrayList<Player>();
        crrtHighestPlyrs.add(plyr1);
```

Game.java

```java
        Menu cntryMenu = new Menu("Which country would you like to relocate too
        int cntryChoice = cntryMenu.getUserChoice();

        plyr.moneyOut(prdctToRelocate.getCostOfRelocation());

        System.out.println();
        System.out.println("£" + prdctToRelocate.getCostOfRelocation() + " has

        prdctToRelocate.addCountry(validCntrys[cntryChoice-1]);
    }

    /**
     * This method gets all the products which the user can relocate
     * @author                  & AdamLogan
     * @param plyr - the current player
     * @param allCmps - all the companies within the game
     * @return
     */
    public static ArrayList<Product> prdctsToRelocate(Player plyr, Company[] al
        ArrayList<Company> cmpnsOwned = companiesOwned(plyr, allCmps);
        ArrayList<Company> validCmpns = new ArrayList<Company>();

        ArrayList<Product> validPrdcts = new ArrayList<Product>();

        // An array of all the valid counties
        Country[] allCntrys = {Country.UK, Country.USA, Country.CANADA,
                Country.FRANCE, Country.GERMANY, Country.IRELAND,
                Country.ITALY, Country.JAPAN};

        for(Company cmp: cmpnsOwned) {
            int prdctsMaxWrhs = 0;
            Product[] prdctsInCmp = cmp.getPrdcts();
            for(Product prdct: prdctsInCmp) {
                if(prdct.getWarehouseCosts().length == prdct.getWarehousesBough
                    prdctsMaxWrhs++;
                }
            }

            if(prdctsMaxWrhs == prdctsInCmp.length) {
                validCmpns.add(cmp);
                for(Product prdct: prdctsInCmp) {
                    if(prdct.getCostOfRelocation() <= plyr.getBalance() &&
                            prdct.getCountriesOwned().length < allCntrys.length
                        validPrdcts.add(prdct);
                    }
                }
            }
        }

        return validPrdcts;
    }
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	15.6 %	1,041	5,638	6,679
src	12.2 %	644	4,642	5,286
fullGame	12.2 %	644	4,642	5,286
Game.java	8.7 %	332	3,500	3,832
Player.java	12.3 %	41	293	334
Product.java	35.0 %	106	197	303
Menu.java	0.0 %	0	156	156
StockExchange.java	0.0 %	0	132	132
Company.java	8.5 %	12	129	141
ProductBidding.java	0.0 %	0	85	85
Square.java	23.9 %	17	54	71
AntiTrust.java	0.0 %	0	49	49
StartOfQuarter.java	0.0 %	0	41	41
Country.java	95.8 %	136	6	142
testing	28.5 %	397	996	1,393
test	28.5 %	397	996	1,393
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.jav	0.0 %	0	192	192
ProductTest.java	0.0 %	0	119	119
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
GameTest.java	99.0 %	397	4	401

Square Test Coverage

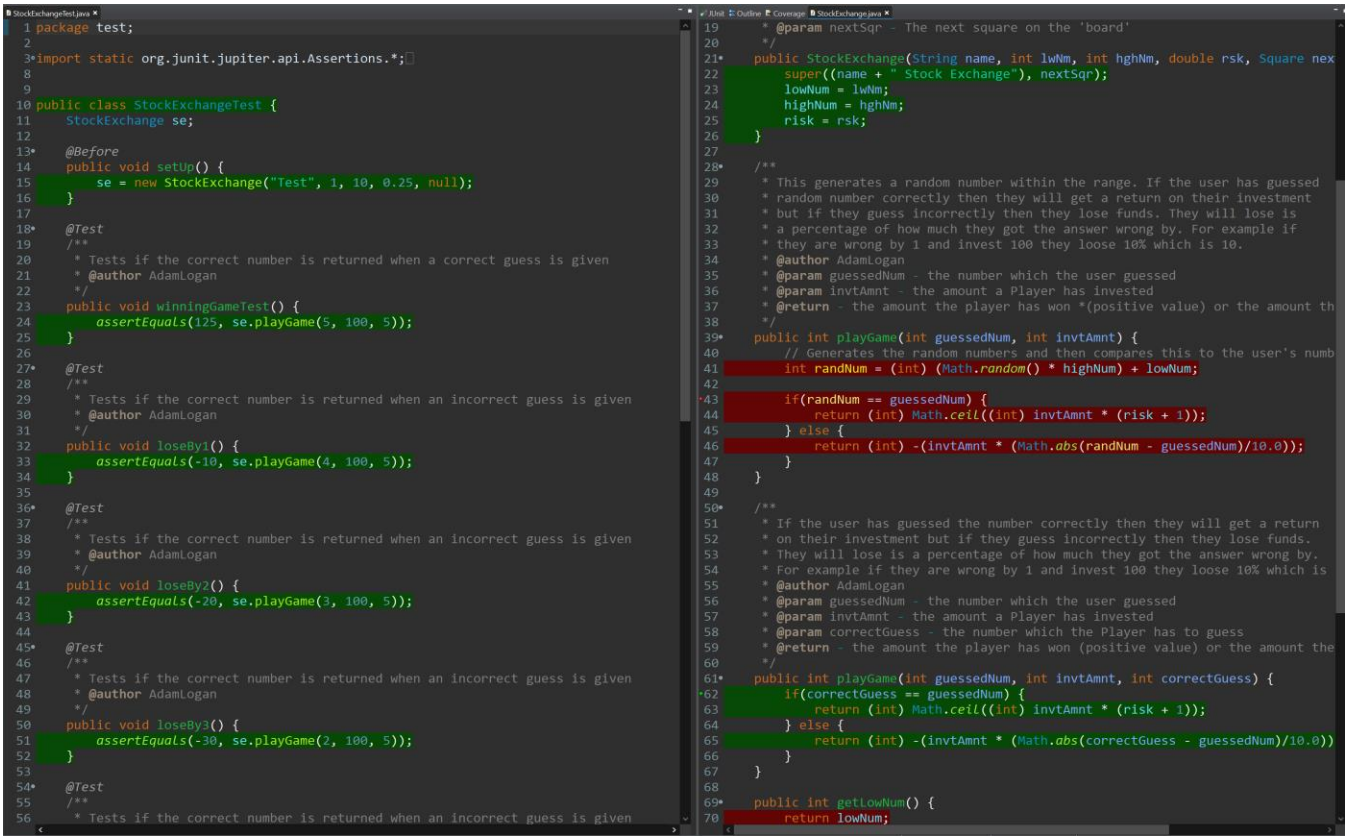


```java
package test;

import static org.junit.jupiter.api.Assertions.*;

public class SquareTest {
    Square sqr;

    @Before
    public void setUp() throws Exception {
        sqr = new Square("Test Square", null);
    }

    @Test
    /**
     * Tests if two Square's with the same names are equal
     * @author AdamLogan
     */
    public void sameSqrTest() {
        Square sqr2 = new Square("Test Square", null);
        assertTrue(sqr.equals(sqr2));
    }

    @Test
    /**
     * Tests if two Square's with the different names are not equal
     * @author AdamLogan
     */
    public void differentSqrTest() {
        Square sqr2 = new Square("Test Square 2", null);
        assertFalse(sqr.equals(sqr2));
    }
}
```

```java
    /**
     * This is the constructor for the Square class
     * @author AdamLogan
     * @param sqrName - The name of the Square
     * @param nextSqr - The next square on the 'board'
     */
    public Square(String sqrName, Square nextSqr) {
        setNextSquare(nextSqr);
        setName(sqrName);
    }

    public Square getNextSquare() {
        return nextSquare;
    }

    public void setNextSquare(Square nextSquare) {
        this.nextSquare = nextSquare;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    /**
     * This checks if this square is the same as another square
     * by checking if the name of the squares are the same
     * @author AdamLogan
     * @param sqr - The other square which is being checked
     * @return - true if the same and false otherwise
     */
    public boolean equals(Square sqr) {
        if(this.getName().equals(sqr.getName())) {
            return true;
        }
        return false;
    }

    public String toString() {
        String output = "";

        String nxtSqrMssg = "";
        if(nextSquare == null) {
            nxtSqrMssg = "There is no next square.";
        } else {
            nxtSqrMssg = "The next square is " + nextSquare.getName();
        }
        output += "The square's name is " + this.name + "\n"
                + nxtSqrMssg;
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	1.0 %	65	6,614	6,679
src	0.6 %	30	5,256	5,286
fullGame	0.6 %	30	5,256	5,286
Game.java	0.0 %	0	3,832	3,832
Player.java	0.0 %	0	334	334
Product.java	0.0 %	0	303	303
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
Company.java	0.0 %	0	141	141
StockExchange.java	0.0 %	0	132	132
ProductBidding.java	0.0 %	0	85	85
AntiTrust.java	0.0 %	0	49	49
Square.java	42.3 %	30	41	71
StartOfQuarter.java	0.0 %	0	41	41
testing	2.5 %	35	1,358	1,393
test	2.5 %	35	1,358	1,393
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.jav	0.0 %	0	192	192
ProductTest.java	0.0 %	0	119	119
StockExchangeTest.java	0.0 %	0	104	104
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
SquareTest.java	100.0 %	35	0	35

StockExchange Test Coverage

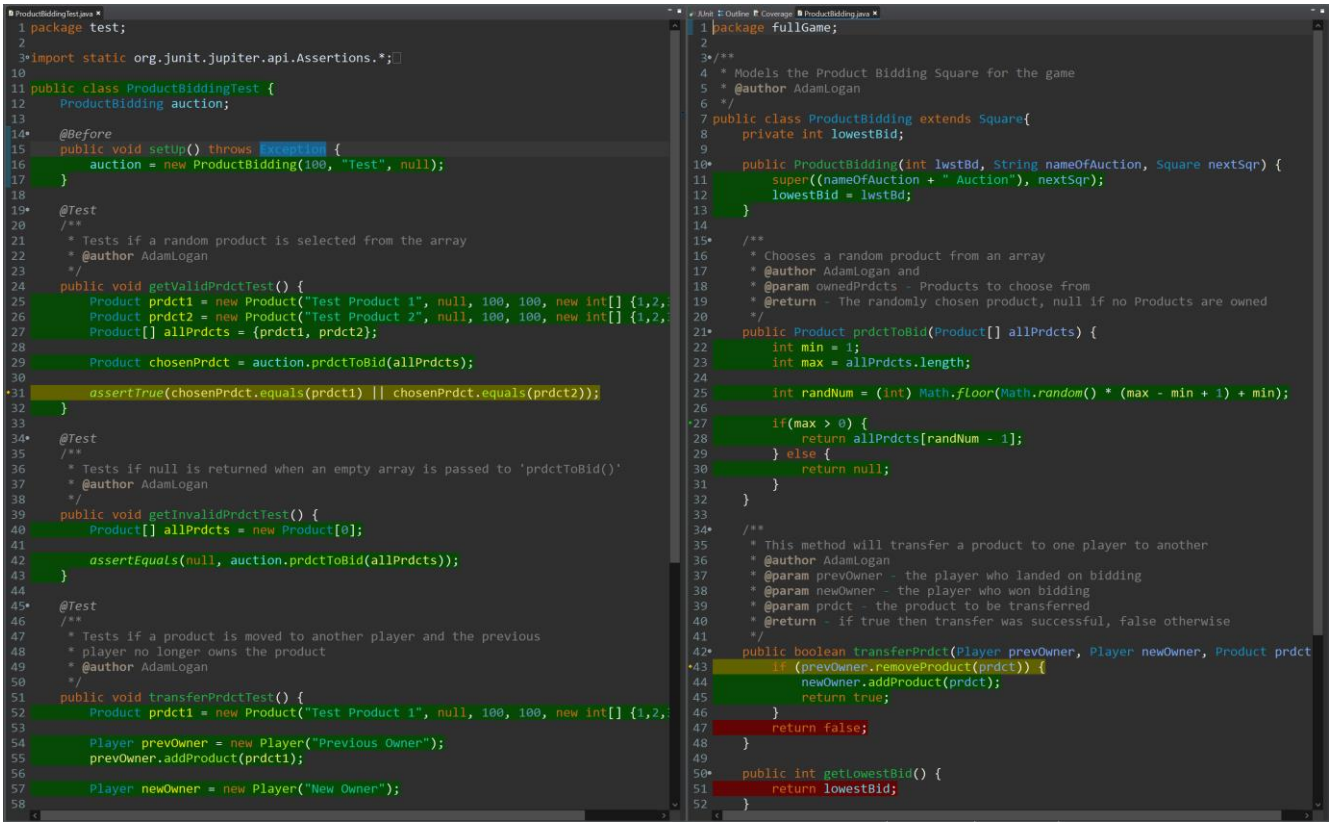


```java
package test;

import static org.junit.jupiter.api.Assertions.*;


public class StockExchangeTest {
    StockExchange se;

    @Before
    public void setUp() {
        se = new StockExchange("Test", 1, 10, 0.25, null);
    }

    @Test
    /**
     * Tests if the correct number is returned when a correct guess is given
     * @author AdamLogan
     */
    public void winningGameTest() {
        assertEquals(125, se.playGame(5, 100, 5));
    }

    @Test
    /**
     * Tests if the correct number is returned when an incorrect guess is given
     * @author AdamLogan
     */
    public void loseBy1() {
        assertEquals(-10, se.playGame(4, 100, 5));
    }

    @Test
    /**
     * Tests if the correct number is returned when an incorrect guess is given
     * @author AdamLogan
     */
    public void loseBy2() {
        assertEquals(-20, se.playGame(3, 100, 5));
    }

    @Test
    /**
     * Tests if the correct number is returned when an incorrect guess is given
     * @author AdamLogan
     */
    public void loseBy3() {
        assertEquals(-30, se.playGame(2, 100, 5));
    }

    @Test
    /**
     * Tests if the correct number is returned when an incorrect guess is given
```

```java
     * @param nextSqr - The next square on the 'board'
     */
    public StockExchange(String name, int lwNm, int hghNm, double rsk, Square nex
        super((name + " Stock Exchange"), nextSqr);
        lowNum = lwNm;
        highNum = hghNm;
        risk = rsk;
    }

    /**
     * This generates a random number within the range. If the user has guessed
     * random number correctly then they will get a return on their investment
     * but if they guess incorrectly then they lose funds. They will lose is
     * a percentage of how much they got the answer wrong by. For example if
     * they are wrong by 1 and invest 100 they loose 10% which is 10.
     * @author AdamLogan
     * @param guessedNum - the number which the user guessed
     * @param invtAmnt - the amount a Player has invested
     * @return - the amount the player has won *(positive value) or the amount th
     */
    public int playGame(int guessedNum, int invtAmnt) {
        // Generates the random numbers and then compares this to the user's numb
        int randNum = (int) (Math.random() * highNum) + lowNum;

        if(randNum == guessedNum) {
            return (int) Math.ceil((int) invtAmnt * (risk + 1));
        } else {
            return (int) -(invtAmnt * (Math.abs(randNum - guessedNum)/10.0));
        }
    }

    /**
     * If the user has guessed the number correctly then they will get a return
     * on their investment but if they guess incorrectly then they lose funds.
     * They will lose is a percentage of how much they got the answer wrong by.
     * For example if they are wrong by 1 and invest 100 they loose 10% which is
     * @author AdamLogan
     * @param guessedNum - the number which the user guessed
     * @param invtAmnt - the amount a Player has invested
     * @param correctGuess - the number which the Player has to guess
     * @return - the amount the player has won (positive value) or the amount the
     */
    public int playGame(int guessedNum, int invtAmnt, int correctGuess) {
        if(correctGuess == guessedNum) {
            return (int) Math.ceil((int) invtAmnt * (risk + 1));
        } else {
            return (int) -(invtAmnt * (Math.abs(correctGuess - guessedNum)/10.0))
        }
    }

    public int getLowNum() {
        return lowNum;
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	2.5 %	168	6,511	6,679
src	1.2 %	64	5,222	5,286
fullGame	1.2 %	64	5,222	5,286
Game.java	0.0 %	0	3,832	3,832
Player.java	0.0 %	0	334	334
Product.java	0.0 %	0	303	303
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
Company.java	0.0 %	0	141	141
ProductBidding.java	0.0 %	0	85	85
StockExchange.java	35.6 %	47	85	132
Square.java	23.9 %	17	54	71
AntiTrust.java	0.0 %	0	49	49
StartOfQuarter.java	0.0 %	0	41	41
testing	7.5 %	104	1,289	1,393
test	7.5 %	104	1,289	1,393
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.jav	0.0 %	0	192	192
ProductTest.java	0.0 %	0	119	119
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
StockExchangeTest.java	100.0 %	104	0	104

ProductBidding Test Coverage

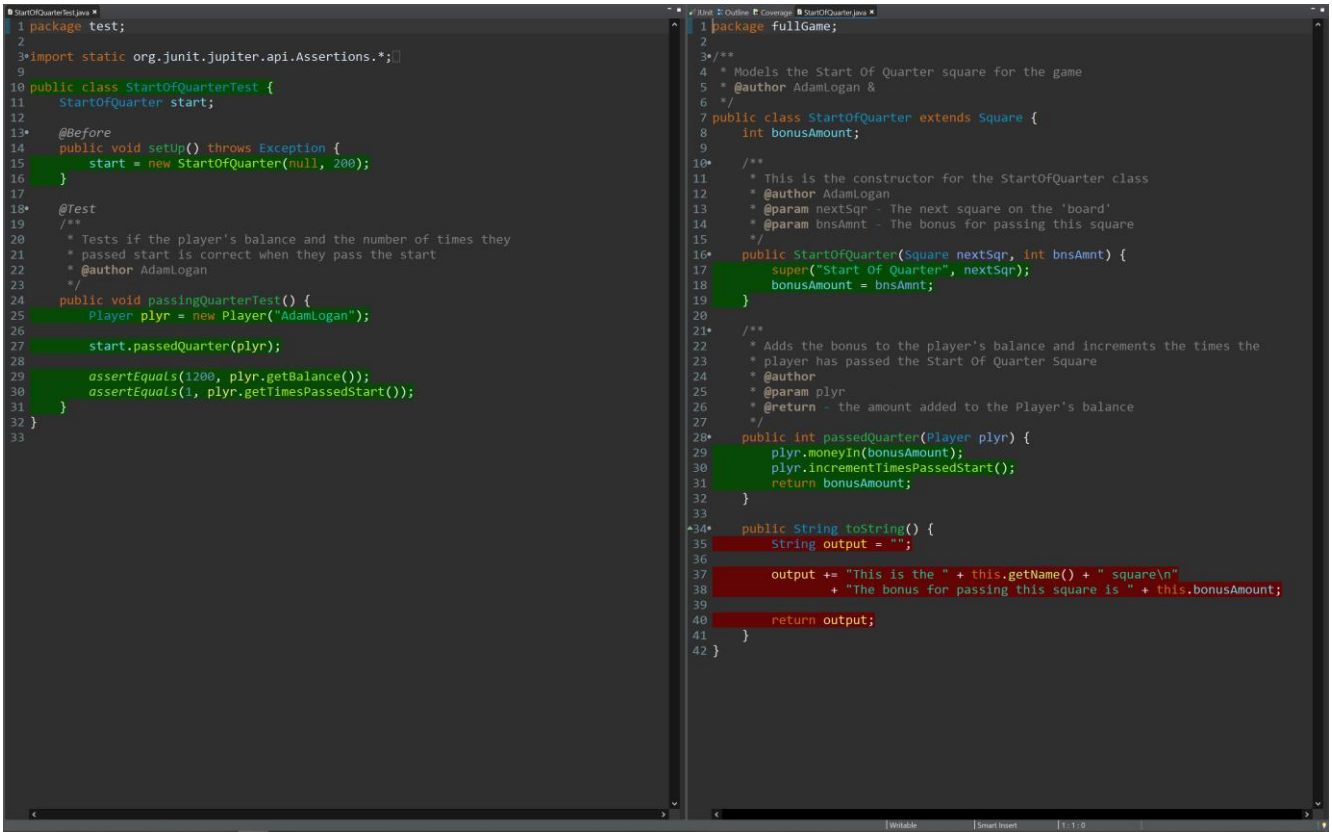


```java
package test;

import static org.junit.jupiter.api.Assertions.*;

public class ProductBiddingTest {
    ProductBidding auction;

    @Before
    public void setUp() throws Exception {
        auction = new ProductBidding(100, "Test", null);
    }

    @Test
    /**
     * Tests if a random product is selected from the array
     * @author AdamLogan
     */
    public void getValidPrdctTest() {
        Product prdct1 = new Product("Test Product 1", null, 100, 100, new int[] {1,2,
        Product prdct2 = new Product("Test Product 2", null, 100, 100, new int[] {1,2,
        Product[] allPrdcts = {prdct1, prdct2};

        Product chosenPrdct = auction.prdctToBid(allPrdcts);

        assertTrue(chosenPrdct.equals(prdct1) || chosenPrdct.equals(prdct2));
    }

    @Test
    /**
     * Tests if null is returned when an empty array is passed to 'prdctToBid()'
     * @author AdamLogan
     */
    public void getInvalidPrdctTest() {
        Product[] allPrdcts = new Product[0];

        assertEquals(null, auction.prdctToBid(allPrdcts));
    }

    @Test
    /**
     * Tests if a product is moved to another player and the previous
     * player no longer owns the product
     * @author AdamLogan
     */
    public void transferPrdctTest() {
        Product prdct1 = new Product("Test Product 1", null, 100, 100, new int[] {1,2,

        Player prevOwner = new Player("Previous Owner");
        prevOwner.addProduct(prdct1);

        Player newOwner = new Player("New Owner");
```

```java
package fullGame;

/**
 * Models the Product Bidding Square for the game
 * @author AdamLogan
 */
public class ProductBidding extends Square{
    private int lowestBid;

    public ProductBidding(int lwstBd, String nameOfAuction, Square nextSqr) {
        super((nameOfAuction + " Auction"), nextSqr);
        lowestBid = lwstBd;
    }

    /**
     * Chooses a random product from an array
     * @author AdamLogan and
     * @param ownedPrdcts - Products to choose from
     * @return - The randomly chosen product, null if no Products are owned
     */
    public Product prdctToBid(Product[] allPrdcts) {
        int min = 1;
        int max = allPrdcts.length;

        int randNum = (int) Math.floor(Math.random() * (max - min + 1) + min);

        if(max > 0) {
            return allPrdcts[randNum - 1];
        } else {
            return null;
        }
    }

    /**
     * This method will transfer a product to one player to another
     * @author AdamLogan
     * @param prevOwner - the player who landed on bidding
     * @param newOwner - the player who won bidding
     * @param prdct - the product to be transferred
     * @return - if true then transfer was successful, false otherwise
     */
    public boolean transferPrdct(Player prevOwner, Player newOwner, Product prdct
        if (prevOwner.removeProduct(prdct)) {
            newOwner.addProduct(prdct);
            return true;
        }
        return false;
    }

    public int getLowestBid() {
        return lowestBid;
    }
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	6.4 %	429	6,250	6,679
src	4.5 %	240	5,046	5,286
fullGame	4.5 %	240	5,046	5,286
Game.java	0.0 %	0	3,832	3,832
Player.java	24.9 %	83	251	334
Product.java	24.4 %	74	229	303
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
Company.java	0.0 %	0	141	141
StockExchange.java	0.0 %	0	132	132
AntiTrust.java	0.0 %	0	49	49
Square.java	42.3 %	30	41	71
StartOfQuarter.java	0.0 %	0	41	41
ProductBidding.java	62.4 %	53	32	85
testing	13.6 %	189	1,204	1,393
test	13.6 %	189	1,204	1,393
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductTest.java	0.0 %	0	119	119
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
ProductBiddingTest.jav	98.4 %	189	3	192

StartOfQuarter Test Coverage

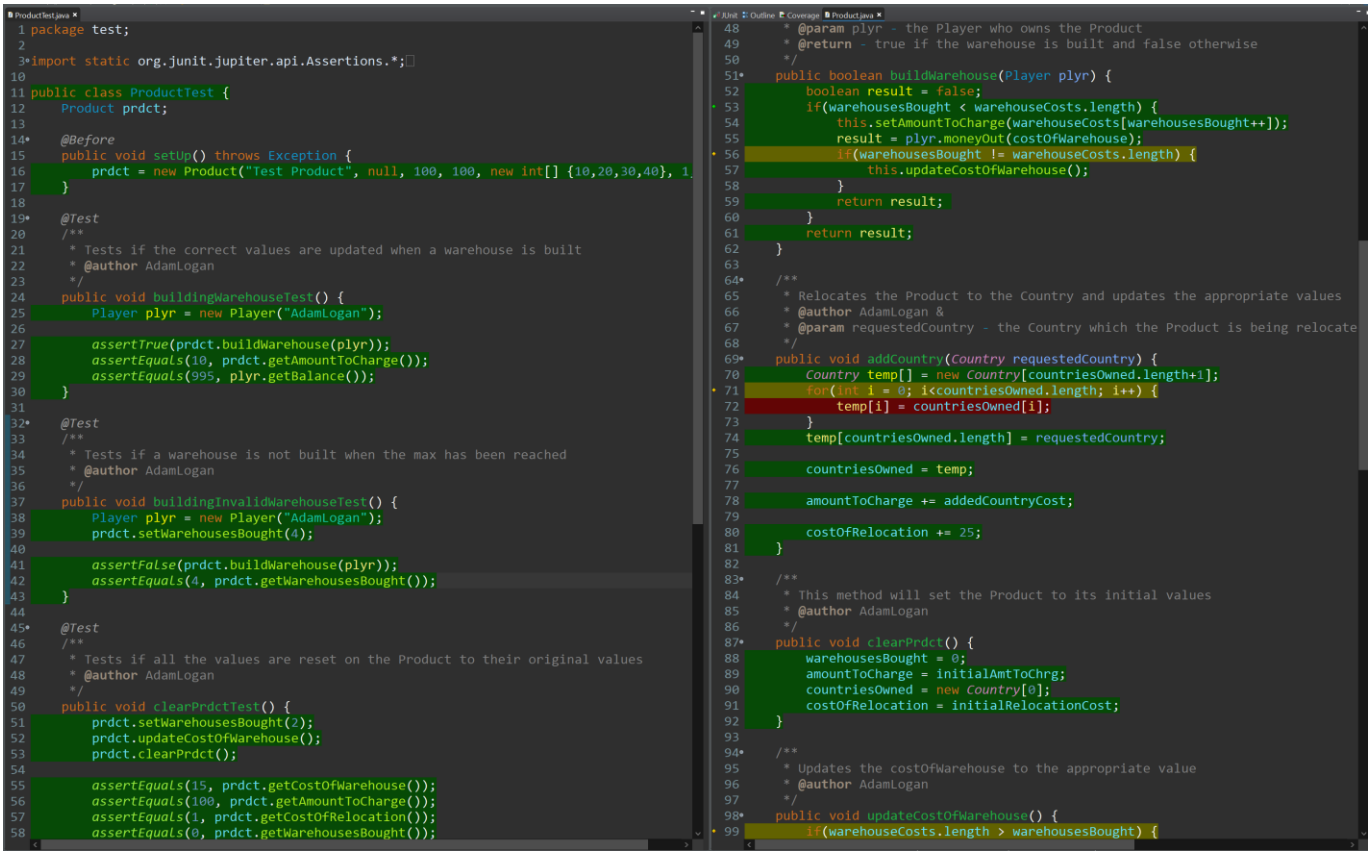


```java
package test;

import static org.junit.jupiter.api.Assertions.*;

public class StartOfQuarterTest {
    StartOfQuarter start;

    @Before
    public void setUp() throws Exception {
        start = new StartOfQuarter(null, 200);
    }

    @Test
    /**
     * Tests if the player's balance and the number of times they
     * passed start is correct when they pass the start
     * @author AdamLogan
     */
    public void passingQuarterTest() {
        Player plyr = new Player("AdamLogan");

        start.passedQuarter(plyr);

        assertEquals(1200, plyr.getBalance());
        assertEquals(1, plyr.getTimesPassedStart());
    }
}
```

```java
package fullGame;

/**
 * Models the Start Of Quarter square for the game
 * @author AdamLogan &
 */
public class StartOfQuarter extends Square {
    int bonusAmount;

    /**
     * This is the constructor for the StartOfQuarter class
     * @author AdamLogan
     * @param nextSqr - The next square on the 'board'
     * @param bnsAmnt - The bonus for passing this square
     */
    public StartOfQuarter(Square nextSqr, int bnsAmnt) {
        super("Start Of Quarter", nextSqr);
        bonusAmount = bnsAmnt;
    }

    /**
     * Adds the bonus to the player's balance and increments the times the
     * player has passed the Start Of Quarter Square
     * @author
     * @param plyr
     * @return - the amount added to the Player's balance
     */
    public int passedQuarter(Player plyr) {
        plyr.moneyIn(bonusAmount);
        plyr.incrementTimesPassedStart();
        return bonusAmount;
    }

    public String toString() {
        String output = "";

        output += "This is the " + this.getName() + " square\n"
                + "The bonus for passing this square is " + this.bonusAmount;

        return output;
    }
}
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	1.6 %	107	6,572	6,679
src	1.5 %	77	5,209	5,286
fullGame	1.5 %	77	5,209	5,286
Game.java	0.0 %	0	3,832	3,832
Product.java	0.0 %	0	303	303
Player.java	12.6 %	42	292	334
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
Company.java	0.0 %	0	141	141
StockExchange.java	0.0 %	0	132	132
ProductBidding.java	0.0 %	0	85	85
Square.java	23.9 %	17	54	71
AntiTrust.java	0.0 %	0	49	49
StartOfQuarter.java	43.9 %	18	23	41
testing	2.2 %	30	1,363	1,393
test	2.2 %	30	1,363	1,393
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.java	0.0 %	0	192	192
ProductTest.java	0.0 %	0	119	119
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
AntiTrustTest.java	0.0 %	0	26	26
StartOfQuarterTest.java	100.0 %	30	0	30

Product Test Coverage

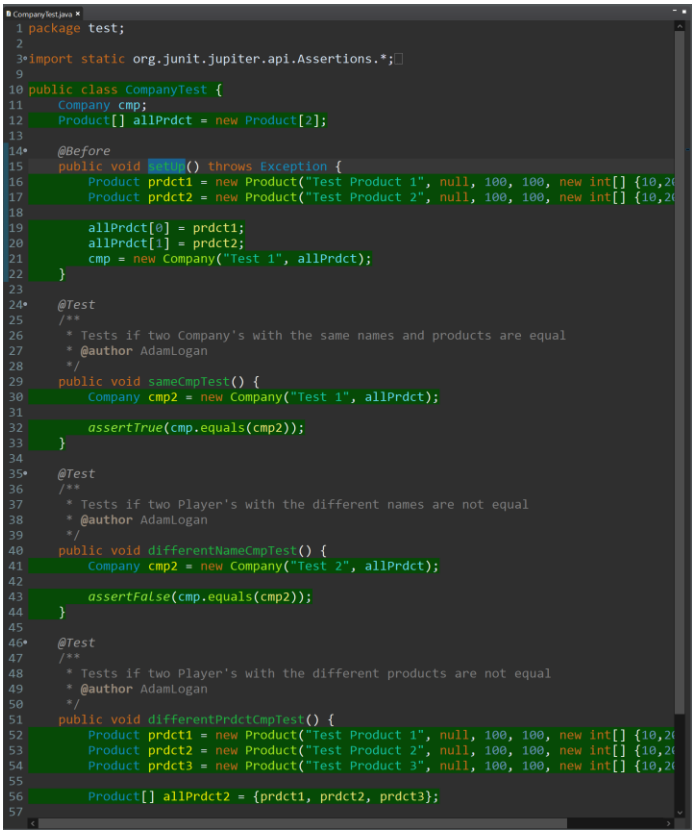


```
1  package test;
2
3  import static org.junit.jupiter.api.Assertions.*;
10
11 public class ProductTest {
12     Product prdct;
13
14     @Before
15     public void setUp() throws Exception {
16         prdct = new Product("Test Product", null, 100, 100, new int[] {10,20,30,40}, 1
17     }
18
19     @Test
20     /**
21      * Tests if the correct values are updated when a warehouse is built
22      * @author AdamLogan
23      */
24     public void buildingWarehouseTest() {
25         Player plyr = new Player("AdamLogan");
26
27         assertTrue(prdct.buildWarehouse(plyr));
28         assertEquals(10, prdct.getAmountToCharge());
29         assertEquals(995, plyr.getBalance());
30     }
31
32     @Test
33     /**
34      * Tests if a warehouse is not built when the max has been reached
35      * @author AdamLogan
36      */
37     public void buildingInvalidWarehouseTest() {
38         Player plyr = new Player("AdamLogan");
39         prdct.setWarehousesBought(4);
40
41         assertFalse(prdct.buildWarehouse(plyr));
42         assertEquals(4, prdct.getWarehousesBought());
43     }
44
45     @Test
46     /**
47      * Tests if all the values are reset on the Product to their original values
48      * @author AdamLogan
49      */
50     public void clearPrdctTest() {
51         prdct.setWarehousesBought(2);
52         prdct.updateCostOfWarehouse();
53         prdct.clearPrdct();
54
55         assertEquals(15, prdct.getCostOfWarehouse());
56         assertEquals(100, prdct.getAmountToCharge());
57         assertEquals(1, prdct.getCostOfRelocation());
58         assertEquals(0, prdct.getWarehousesBought());
```

```
48      * @param plyr - the Player who owns the Product
49      * @return - true if the warehouse is built and false otherwise
50      */
51     public boolean buildWarehouse(Player plyr) {
52         boolean result = false;
53         if(warehousesBought < warehouseCosts.length) {
54             this.setAmountToCharge(warehouseCosts[warehousesBought++]);
55             result = plyr.moneyOut(costOfWarehouse);
56             if(warehousesBought != warehouseCosts.length) {
57                 this.updateCostOfWarehouse();
58             }
59             return result;
60         }
61         return result;
62     }
63
64     /**
65      * Relocates the Product to the Country and updates the appropriate values
66      * @author AdamLogan &
67      * @param requestedCountry - the Country which the Product is being relocate
68      */
69     public void addCountry(Country requestedCountry) {
70         Country temp[] = new Country[countriesOwned.length+1];
71         for(int i = 0; i<countriesOwned.length; i++) {
72             temp[i] = countriesOwned[i];
73         }
74         temp[countriesOwned.length] = requestedCountry;
75
76         countriesOwned = temp;
77
78         amountToCharge += addedCountryCost;
79
80         costOfRelocation += 25;
81     }
82
83     /**
84      * This method will set the Product to its initial values
85      * @author AdamLogan
86      */
87     public void clearPrdct() {
88         warehousesBought = 0;
89         amountToCharge = initialAmtToChrg;
90         countriesOwned = new Country[0];
91         costOfRelocation = initialRelocationCost;
92     }
93
94     /**
95      * Updates the costOfWarehouse to the appropriate value
96      * @author AdamLogan
97      */
98     public void updateCostOfWarehouse() {
99         if(warehouseCosts.length > warehousesBought) {
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	7.7 %	518	6,189	6,707
src	7.1 %	374	4,912	5,286
fullGame	7.1 %	374	4,912	5,286
Game.java	0.0 %	0	3,832	3,832
Player.java	9.9 %	33	301	334
Menu.java	0.0 %	0	156	156
Company.java	0.0 %	0	141	141
StockExchange.java	0.0 %	0	132	132
Product.java	62.0 %	188	115	303
ProductBidding.java	0.0 %	0	85	85
Square.java	23.9 %	17	54	71
AntiTrust.java	0.0 %	0	49	49
StartOfQuarter.java	0.0 %	0	41	41
Country.java	95.8 %	136	6	142
testing	10.1 %	144	1,277	1,421
test	10.1 %	144	1,277	1,421
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.java	0.0 %	0	192	192
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
ProductTest.java	98.0 %	144	3	147

Company Test Coverage

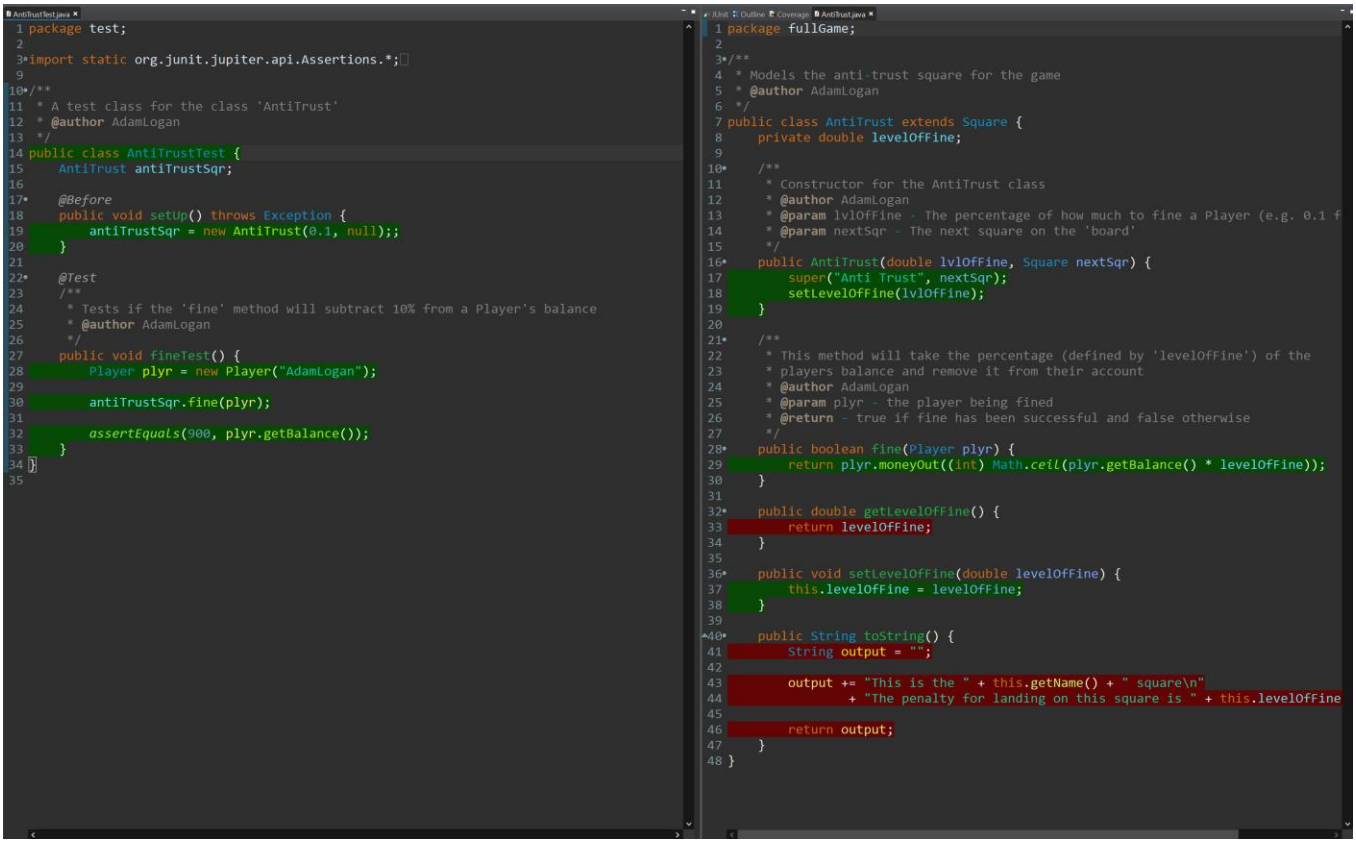


```
1  package test;
2
3  import static org.junit.jupiter.api.Assertions.*;
9
10 public class CompanyTest {
11     Company cmp;
12     Product[] allPrdct = new Product[2];
13
14     @Before
15     public void setUp() throws Exception {
16         Product prdct1 = new Product("Test Product 1", null, 100, 100, new int[] {10,20
17         Product prdct2 = new Product("Test Product 2", null, 100, 100, new int[] {10,20
18
19         allPrdct[0] = prdct1;
20         allPrdct[1] = prdct2;
21         cmp = new Company("Test 1", allPrdct);
22     }
23
24     @Test
25     /**
26      * Tests if two Company's with the same names and products are equal
27      * @author AdamLogan
28      */
29     public void sameCmpTest() {
30         Company cmp2 = new Company("Test 1", allPrdct);
31
32         assertTrue(cmp.equals(cmp2));
33     }
34
35     @Test
36     /**
37      * Tests if two Player's with the different names are not equal
38      * @author AdamLogan
39      */
40     public void differentNameCmpTest() {
41         Company cmp2 = new Company("Test 2", allPrdct);
42
43         assertFalse(cmp.equals(cmp2));
44     }
45
46     @Test
47     /**
48      * Tests if two Player's with the different products are not equal
49      * @author AdamLogan
50      */
51     public void differentPrdctCmpTest() {
52         Product prdct1 = new Product("Test Product 1", null, 100, 100, new int[] {10,20
53         Product prdct2 = new Product("Test Product 2", null, 100, 100, new int[] {10,20
54         Product prdct3 = new Product("Test Product 3", null, 100, 100, new int[] {10,20
55
56         Product[] allPrdct2 = {prdct1, prdct2, prdct3};
57
```

```
23     }
24
25     public String getName() {
26         return name;
27     }
28
29     public String toString() {
30         String output = "";
31
32         output += "The comapnay's name is " + this.getName() + "\n";
33
34         if(this.prdcts.length > 0) {
35             output += "Products in company: \n";
36             for(Product prdct: prdcts) {
37                 output += "-" + prdct.getName() + "\n";
38             }
39         }
40
41         return output;
42     }
43
44     /**
45      * Checks if this company object is the same as another Company object,
46      * which they are equal if the name is the same and they have the same Produc
47      * @param cmp - the Company that is being checked
48      * @return - true if equal and false otherwise
49      */
50     public boolean equals(Company cmp) {
51         boolean result = false;
52
53         if(!this.name.equals(cmp.getName())) {
54             return false;
55         }
56
57         for(Product crrntPrdct: this.prdcts) {
58             for(Product othrPrdct: cmp.getPrdcts()) {
59                 if(crrntPrdct.equals(othrPrdct)) {
60                     result = true;
61                     break;
62                 } else {
63                     result = false;
64                 }
65             }
66         }
67
68         if(this.prdcts.length != cmp.getPrdcts().length) {
69             result = false;
70         }
71
72         return result;
73     }
74 }
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	5.8 %	386	6,321	6,707
src	3.2 %	167	5,119	5,286
fullGame	3.2 %	167	5,119	5,286
Game.java	0.0 %	0	3,832	3,832
Player.java	0.0 %	0	334	334
Product.java	19.1 %	58	245	303
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
StockExchange.java	0.0 %	0	132	132
ProductBidding.java	0.0 %	0	85	85
Company.java	56.0 %	79	62	141
AntiTrust.java	0.0 %	0	49	49
Square.java	42.3 %	30	41	71
StartOfQuarter.java	0.0 %	0	41	41
testing	15.4 %	219	1,202	1,421
test	15.4 %	219	1,202	1,421
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
ProductBiddingTest.java	0.0 %	0	192	192
ProductTest.java	0.0 %	0	147	147
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
CompanyTest.java	100.0 %	219	0	219

AntiTrust Test Coverage

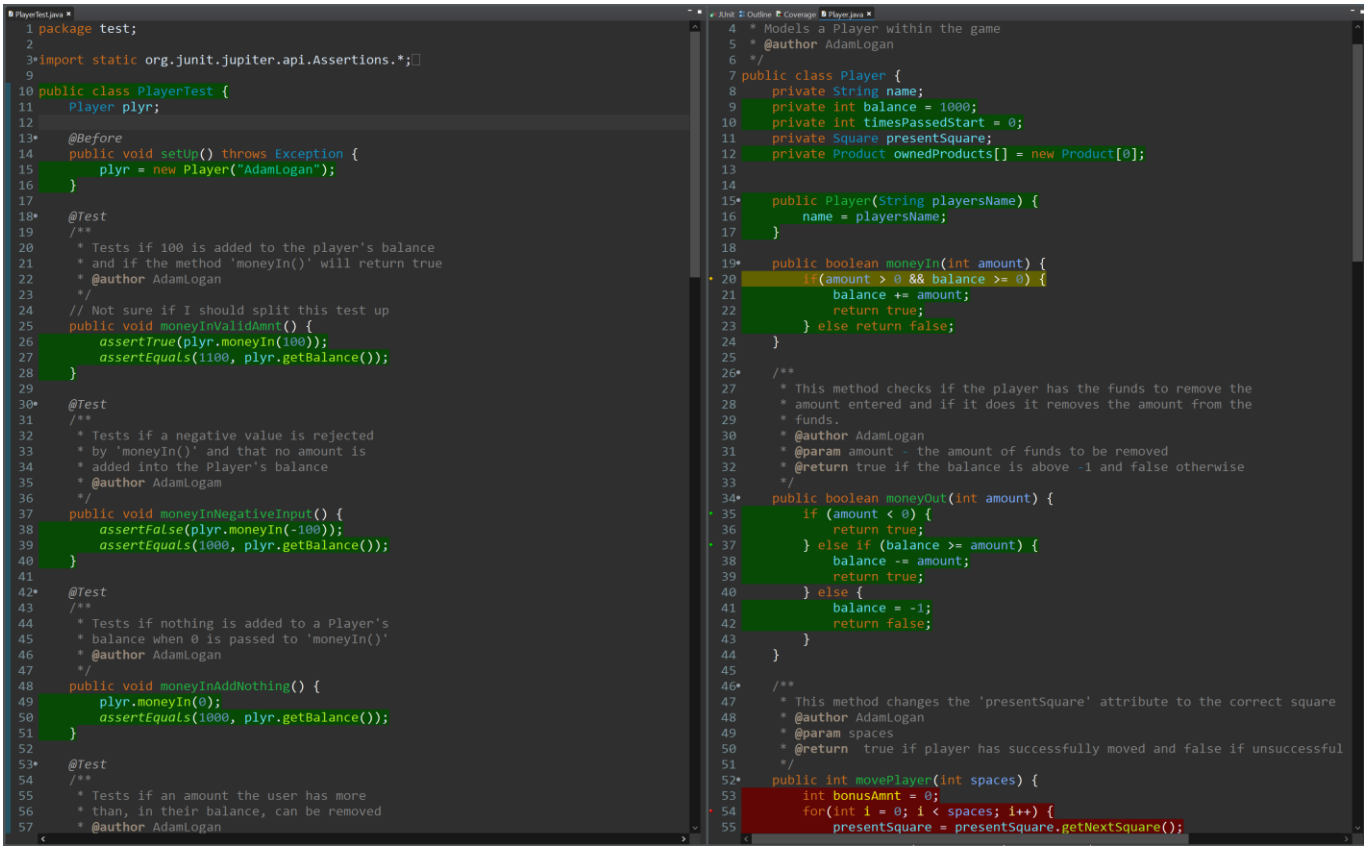


```java
package test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * A test class for the class 'AntiTrust'
 * @author AdamLogan
 */
public class AntiTrustTest {
    AntiTrust antiTrustSqr;

    @Before
    public void setUp() throws Exception {
        antiTrustSqr = new AntiTrust(0.1, null);;
    }

    @Test
    /**
     * Tests if the 'fine' method will subtract 10% from a Player's balance
     * @author AdamLogan
     */
    public void fineTest() {
        Player plyr = new Player("AdamLogan");

        antiTrustSqr.fine(plyr);

        assertEquals(900, plyr.getBalance());
    }
}
```

```java
package fullGame;

/**
 * Models the anti-trust square for the game
 * @author AdamLogan
 */
public class AntiTrust extends Square {
    private double levelOfFine;

    /**
     * Constructor for the AntiTrust class
     * @author AdamLogan
     * @param lvlOfFine - The percentage of how much to fine a Player (e.g. 0.1 f
     * @param nextSqr - The next square on the 'board'
     */
    public AntiTrust(double lvlOfFine, Square nextSqr) {
        super("Anti Trust", nextSqr);
        setLevelOfFine(lvlOfFine);
    }

    /**
     * This method will take the percentage (defined by 'levelOfFine') of the
     * players balance and remove it from their account
     * @author AdamLogan
     * @param plyr - the player being fined
     * @return - true if fine has been successful and false otherwise
     */
    public boolean fine(Player plyr) {
        return plyr.moneyOut((int) Math.ceil(plyr.getBalance() * levelOfFine));
    }

    public double getLevelOfFine() {
        return levelOfFine;
    }

    public void setLevelOfFine(double levelOfFine) {
        this.levelOfFine = levelOfFine;
    }

    public String toString() {
        String output = "";

        output += "This is the " + this.getName() + " square\n"
                + "The penalty for landing on this square is " + this.levelOfFine

        return output;
    }
}
```

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	1.5 %	99	6,608	6,707
src	1.4 %	73	5,213	5,286
fullGame	1.4 %	73	5,213	5,286
Game.java	0.0 %	0	3,832	3,832
Product.java	0.0 %	0	303	303
Player.java	9.9 %	33	301	334
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
Company.java	0.0 %	0	141	141
StockExchange.java	0.0 %	0	132	132
ProductBidding.java	0.0 %	0	85	85
Square.java	23.9 %	17	54	71
StartOfQuarter.java	0.0 %	0	41	41
AntiTrust.java	46.9 %	23	26	49
testing	1.8 %	26	1,395	1,421
test	1.8 %	26	1,395	1,421
GameTest.java	0.0 %	0	401	401
PlayerTest.java	0.0 %	0	267	267
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.java	0.0 %	0	192	192
ProductTest.java	0.0 %	0	147	147
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	100.0 %	26	0	26

Player Test Coverage



```java
package test;

import static org.junit.jupiter.api.Assertions.*;

public class PlayerTest {
    Player plyr;

    @Before
    public void setUp() throws Exception {
        plyr = new Player("AdamLogan");
    }

    @Test
    /**
     * Tests if 100 is added to the player's balance
     * and if the method 'moneyIn()' will return true
     * @author AdamLogan
     */
    // Not sure if I should split this test up
    public void moneyInValidAmnt() {
        assertTrue(plyr.moneyIn(100));
        assertEquals(1100, plyr.getBalance());
    }

    @Test
    /**
     * Tests if a negative value is rejected
     * by 'moneyIn()' and that no amount is
     * added into the Player's balance
     * @author AdamLogam
     */
    public void moneyInNegativeInput() {
        assertFalse(plyr.moneyIn(-100));
        assertEquals(1000, plyr.getBalance());
    }

    @Test
    /**
     * Tests if nothing is added to a Player's
     * balance when 0 is passed to 'moneyIn()'
     * @author AdamLogan
     */
    public void moneyInAddNothing() {
        plyr.moneyIn(0);
        assertEquals(1000, plyr.getBalance());
    }

    @Test
    /**
     * Tests if an amount the user has more
     * than, in their balance, can be removed
     * @author AdamLogan
```

```java
/**
 * Models a Player within the game
 * @author AdamLogan
 */
public class Player {
    private String name;
    private int balance = 1000;
    private int timesPassedStart = 0;
    private Square presentSquare;
    private Product ownedProducts[] = new Product[0];


    public Player(String playersName) {
        name = playersName;
    }

    public boolean moneyIn(int amount) {
        if(amount > 0 && balance >= 0) {
            balance += amount;
            return true;
        } else return false;
    }

    /**
     * This method checks if the player has the funds to remove the
     * amount entered and if it does it removes the amount from the
     * funds.
     * @author AdamLogan
     * @param amount - the amount of funds to be removed
     * @return true if the balance is above -1 and false otherwise
     */
    public boolean moneyOut(int amount) {
        if (amount < 0) {
            return true;
        } else if (balance >= amount) {
            balance -= amount;
            return true;
        } else {
            balance = -1;
            return false;
        }
    }

    /**
     * This method changes the 'presentSquare' attribute to the correct square
     * @author AdamLogan
     * @param spaces
     * @return  true if player has successfully moved and false if unsuccessful
     */
    public int movePlayer(int spaces) {
        int bonusAmnt = 0;
        for(int i = 0; i < spaces; i++) {
            presentSquare = presentSquare.getNextSquare();
```

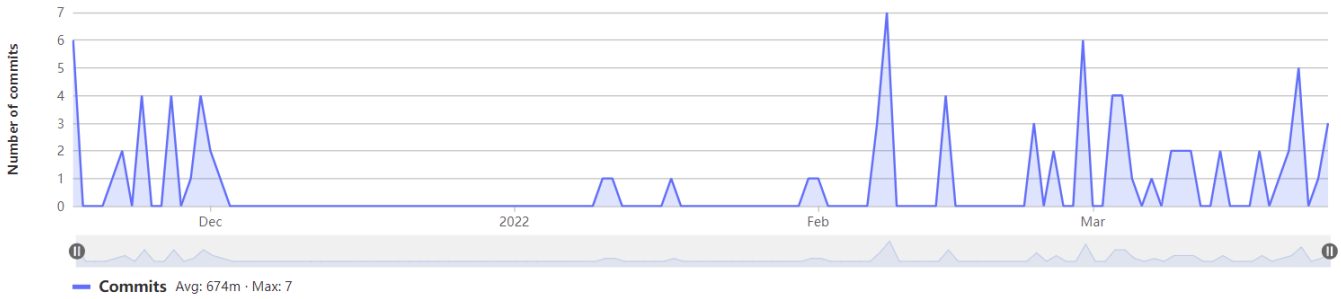
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Technopoly	7.8 %	523	6,184	6,707
src	4.8 %	256	5,030	5,286
fullGame	4.8 %	256	5,030	5,286
Game.java	0.0 %	0	3,832	3,832
Product.java	24.4 %	74	229	303
Player.java	45.5 %	152	182	334
Menu.java	0.0 %	0	156	156
Country.java	0.0 %	0	142	142
Company.java	0.0 %	0	141	141
StockExchange.java	0.0 %	0	132	132
ProductBidding.java	0.0 %	0	85	85
AntiTrust.java	0.0 %	0	49	49
Square.java	42.3 %	30	41	71
StartOfQuarter.java	0.0 %	0	41	41
testing	18.8 %	267	1,154	1,421
test	18.8 %	267	1,154	1,421
GameTest.java	0.0 %	0	401	401
CompanyTest.java	0.0 %	0	219	219
ProductBiddingTest.java	0.0 %	0	192	192
ProductTest.java	0.0 %	0	147	147
StockExchangeTest.java	0.0 %	0	104	104
SquareTest.java	0.0 %	0	35	35
StartOfQuarterTest.java	0.0 %	0	30	30
AntiTrustTest.java	0.0 %	0	26	26
PlayerTest.java	100.0 %	267	0	267

GitLab Activity

GitLab Commit Graphs

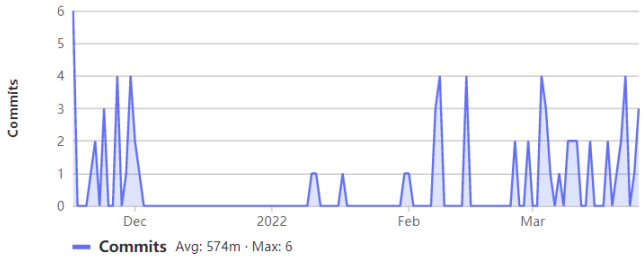
Commits to master

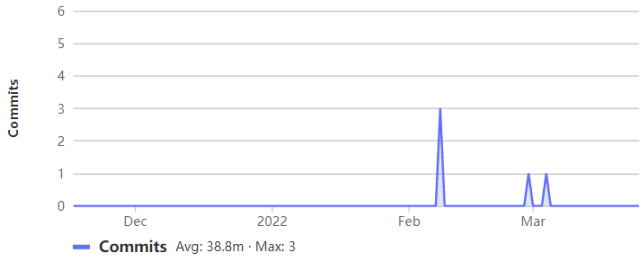
Excluding merge commits. Limited to 6,000 commits.



Adam Logan

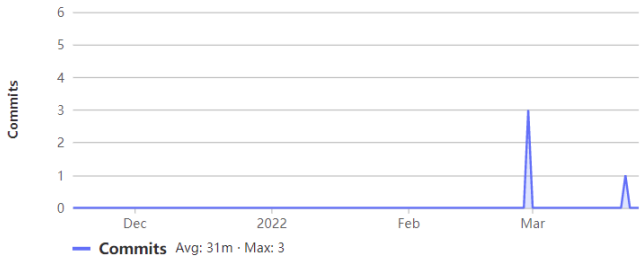
74 commits (alogan20@qub.ac.uk)

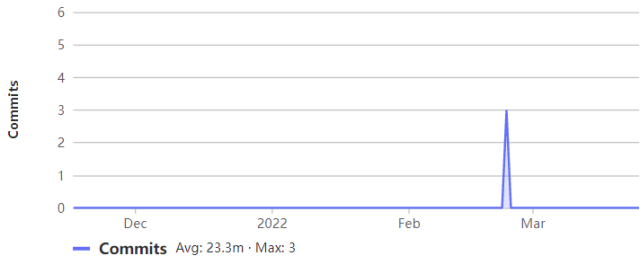




adamlogan17

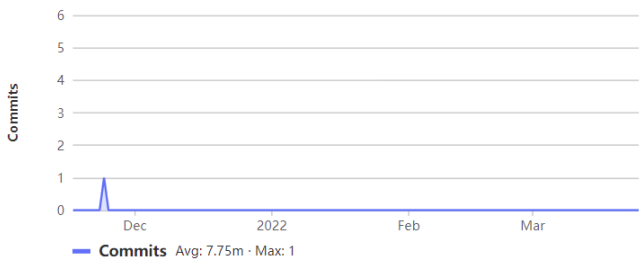
3 commits (55030864+adamlogan17@users.noreply.github.com)





Adam Logan

1 commit (40293585@eeecs.qub.ac.uk)



Select GitLab Commits

Pushed to branch `master`

`54875a7b` · Joe added incomeEvent, valid menu options and passStart method

Pushed to branch `master`

`5354ec99` · Fixed spacing and spelling mistakes

40293585 @40293585

Pushed to branch `master`

`1348d9cf` · along with Jake, implemented requestCountry

40293585 @40293585

Pushed to branch `master`

`2f4fa66c` · added JUnit tests and created an increasing cost for warehouses

40293585 @40293585

Pushed to branch `master`

`28661d44` · Implemented the Stock Exchange Game

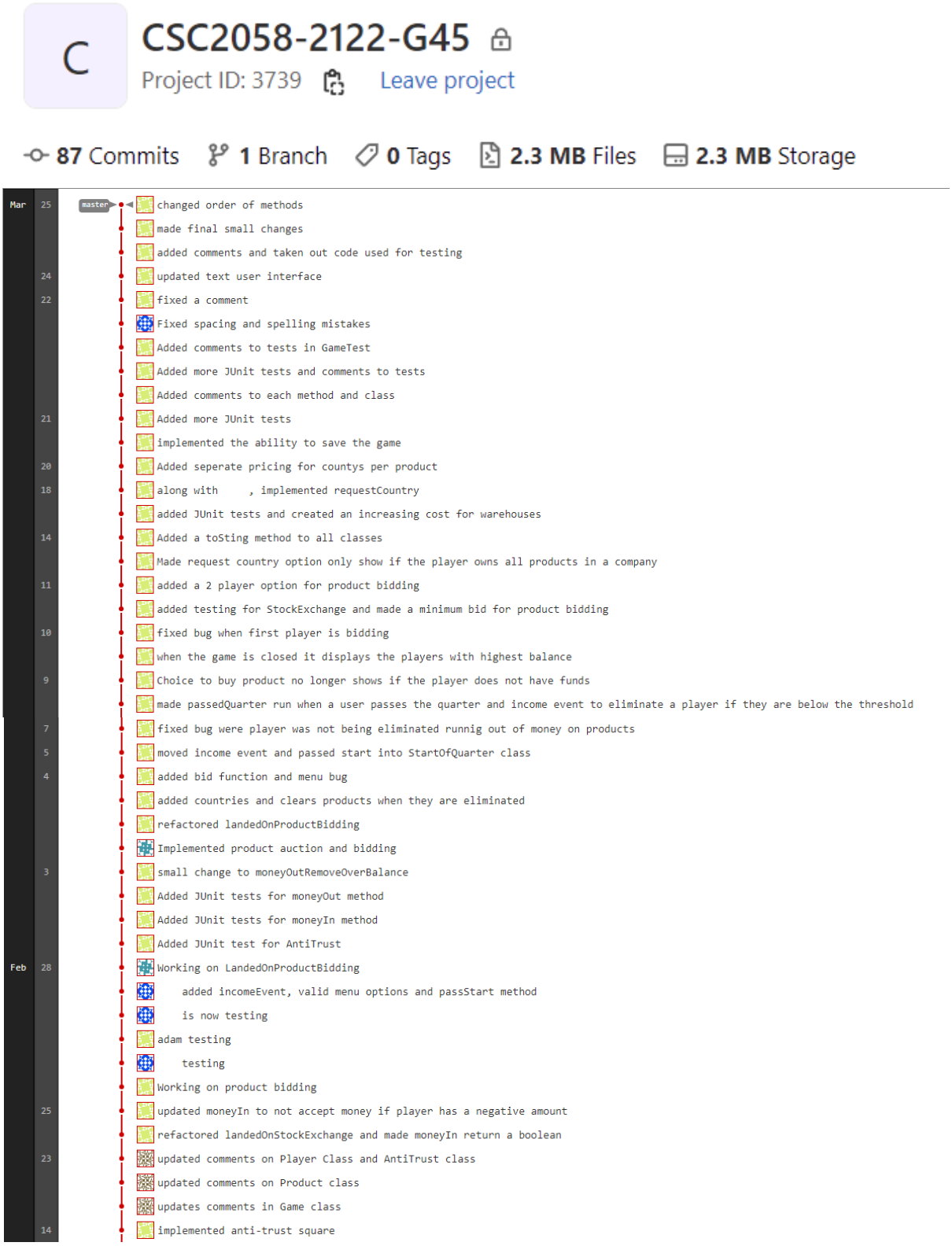
Pushed to branch `master`

`9e6457c0` · Working on LandedOnProductBidding

Pushed to branch `master`

`ba1b1465` · Implemented product auction and bidding

All GitLab Commits



	added getName() back
	few variable name changes
	changes to the Game class
29	changed several classes
27	chnages to Product class
	Delete Square.java
	Delete Square.class
	added code from google drive
24	Update Game.java
	addes square class
	Update Console.java
	Update Game.java
22	Update Game.java
	Created the console, game and menu classes
21	Added document folder
17	another change made in eclipse
	commit made through bash
	Update Test.java - change made on gitlab
	test
	Delete test
	Add new file
	Added functionality to transferPrdct
	created clearPrdct method in the Product class
	created method removeProduct in Player class
8	added message when player has all products of a company
	fixed check inventory and commented out ShaunCathals product bidding
	added contains method and started LandedOnProductBidding
	Check 3
	check 2
	check
	created check for products in company
7	deleted test class
	added inventory
	added test class
1	Implemented the Stock Exchange Game
Jan 31	Completed the implementation for warehouses
17	Added more comments
11	Added comments
10	Added comments to methods
Dec 2	changed a few return types
1	fixed player being charged for product
	added in product names and prices
Nov 30	changed office to warehouse