

## CSC7053 Peer Assessment: ArtemisLite

Evaluation		Group Number:		
Name	Contribution to team-working and motivation <sup>1</sup>	Contribution to documented analysis, design and testing <sup>1,2</sup>	Contribution to working system code <sup>1,2</sup>	Peer Score (Range 85 – 115)
<i>Pierce Doherty</i>	4	3	3	
Jonathan Farrell	4	5	3	
Kyle Carson	5	4	5	

Declaration		
<p>"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."</p>		
Name	Date	Confirmation ( <i>use the words shown in the example below!</i> )
<i>Pierce Doherty</i>	22/04/21	I agree to the terms of the declaration.
Jonathan Farrell	22/04/21	I agree to the terms of the declaration.
Kyle Carson	22/04/21	I agree to the terms of the declaration.

Personal statement of (enter name):	<i>Pierce Doherty</i>
The following were my most significant contributions to the project (100 words or less):	
<p>Throughout the project I was responsible for creating some of the use case descriptions (pay player, take turn), sequences diagrams (take turn, chance square) and general updates on parts of the report as the project progressed. I initially wrote the first iteration of the player class and one or two smaller classes which were updated several times after this by another member of the group. I was also responsible for testing and spotting bugs in the code.</p>	

<i>Personal statement of (enter name):</i>	<i>Jonathan Farrell</i>
The following were my most significant contributions to the project (100 words or less):	
<ul style="list-style-type: none"> <li>- Creation and alteration of the use case descriptions</li> <li>- Creation and alteration of the sequence diagrams</li> <li>- Production of the 'Requirements Analysis', 'Realisation' and 'Design' sections of the report</li> <li>- Development of a section of the 'Game' class and the addition of multiple other required methods to the code</li> <li>- Amendment of segments of the UML Class diagram</li> </ul>	

<i>Personal statement of (enter name):</i>	<i>Kyle Carson</i>
The following were my most significant contributions to the project (100 words or less):	
<ul style="list-style-type: none"> <li>- Constructed the use case diagram and created initial use case descriptions for game set-up and end game</li> <li>- Constructed set-up and end game sequence diagrams</li> <li>- Constructed initial UML Class Diagram</li> <li>- Implemented working functionality of code and assisted in fixing bugs following acceptance testing</li> </ul>	

## CSC 7053 - ArtemisLite Report

---

### **Group 21 – Kyle Carson, Jonathan Farrell, Pierce Doherty**

The aim of this project was to design and develop a system that is able to function as a virtual board game without the use of a graphical user interface. It was proposed that the game should be played using the console of a software development package with text-based user interaction. The game is based on NASA's target to land the first woman and next man on the moon by 2024 with the in-game tasks set out to mirror the mission in a simplified fashion. Allowing 2-4 players to participate, the game involves teamwork and the use of resources to fully develop each of the systems whereby the climax is reached. The player with the highest total of resources used and remaining balance is declared the winner.

Unified Modelling Language (UML) was used to portray the analysis of the requirements with a diagram to also display the design of the system. The 'use case driven' development involved the creation of descriptions for each of the use cases, which are presented in a tabular format. The use case descriptions were used to invoke sequence diagrams which show how the various classes interact with each other during the playing of the game. A UML diagram was produced to provide a visual representation of how each of the classes are linked, with the appropriate methods and variables contained.

There were certain features identified as the customer's key requirements at the outset of the project with an objective to achieve their inclusion within the system. The game was to allow a maximum of 4 competitors, with each taking turns to roll virtual dice and move around the board which can be visualised in the design section. Players were to be allowed to take charge of unowned squares via the exchange of resources, while also being allowed to refuse the purchase. The production has accounted for this requirement with the implementation of an auction where a player has insufficient funds or rejects the option to buy a square. Players are required to pay a fine when they land on a competitors system square. The game also required the inclusion of a 'Start' square to allow their resources to be replenished. 'Blank' squares with no functionality were incorporated as requested by the customer.

The customer demanded that four 'systems' were to be included with two made up of two adjacent squares and two consisting of three adjacent squares. One of the two square systems (Ground Systems) was assigned as the most expensive while the other was made the cheapest (Research Systems) to satisfy requirements. The produced system allows a square to be developed by a player even when they are not positioned on it and they may undertake multiple developments during one turn. A square must have had three single developments completed before a major development can be implemented. When each of the system squares have been fully developed, the game is complete and a representation of the final state of play is output to the user interface.

## Requirements Analysis (Main Author: JF)

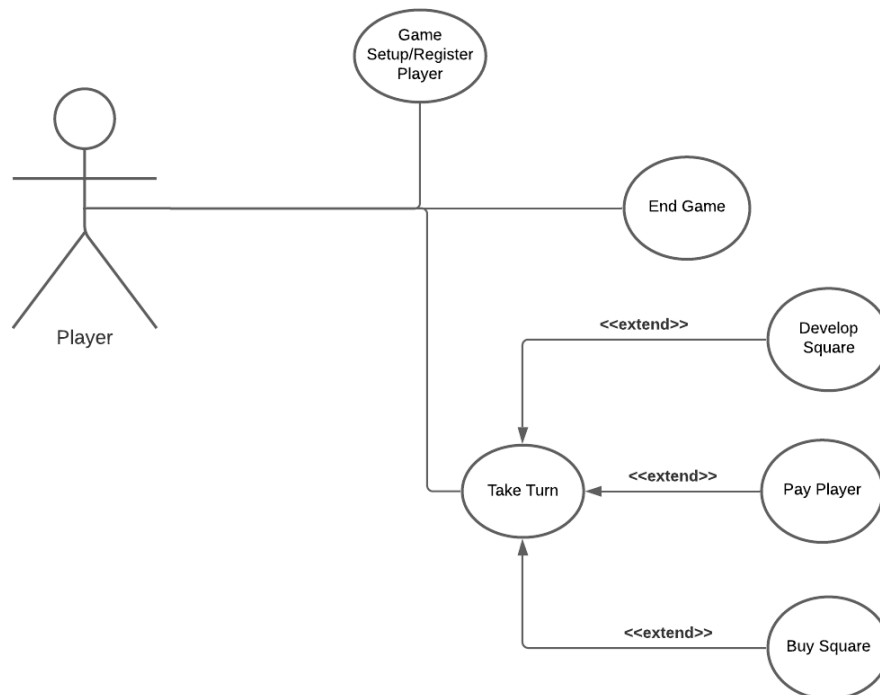


Figure 1: UML Use Case Diagram for the proposed 'ArtemisLite' game.

The UML Use Case Diagram shown above was produced to provide a representation of the entire system with each of the use cases entailing further sequences of actions. Each of the ellipses in the diagram constitutes a use case which have associated descriptions to summarise the events which follow. The 'Game Setup/Register Player' use case is mandatory for the game to be started. This is followed in the flow by the 'Take Turn' use case which extends each of 'Develop Square', 'Pay Player' and 'Buy Square' as these are exclusive during a player's turn. The 'End Game' use case is accessed when the board has been completed or if a player decides to quit.

Use Case	Game Setup/Register Player
Actors	Player
Pre-Condition	-

Main Flow	<ol style="list-style-type: none"> <li>1. A player opens the application, and the screen displays a welcome message and a prompt to enter the number of players (2-4).</li> <li>2. The player enters a number between 2-4 to select the number of participants playing the game.</li> <li>3. The system confirms the number of teams that are being created and prompts the user to enter their team's name.</li> <li>4. The user enters a name for each of the teams and the system creates a player in each case.</li> <li>5. The system confirms that the teams have been created and the game can begin.</li> </ol>
Alternative Flows	<ol style="list-style-type: none"> <li>1. At 4. in the main flow, the user enters a team name that has already been entered. The user is asked to re-enter the team's name.</li> </ol>
Post-Condition	Selected number of teams have been created and the game can begin.

*Figure 2 : Use Case description for 'Game Setup/Register Player'*

The 'Game Setup/Register Player' use case shown above, describes the flow actions a user takes to allow the game to be played. Providing the use case has been executed correctly, the selected number of players will have been registered and the game will be ready to begin. There is an alternative flow associated with entering the team names however there is only a single way that the game can be set up.

The 'Take Turn' use case shown below describes the events that can occur during a player's turn while the game is being played. The dice roll is mandatory and is part of the main flow, however there are various alternative flows depending on the type of square the player lands on. These alternative flows involve the extension of other use cases where appropriate including 'Buy Square', 'Pay Player' and 'Develop Square'. The player will be presented with options depending on their situation on the board and if they input their selections correctly it will be the next player's turn, signifying a successful outcome.

Use Case	Take Turn
Actors	Player
Pre-conditions	Player is registered and ready to play.
Main Flow	<ol style="list-style-type: none"> <li>1. System prompts the player with a message if they want to roll the dice.</li> <li>2. User accepts.</li> <li>3. System moves the player to the correct square given the sum of the two dice.</li> <li>4. If the player passes 'GO' (also know as Space Systems HQ) their balance is updated.</li> <li>5. System identifies the square and presents the user with options.</li> </ol>
Alternative Flow	<ol style="list-style-type: none"> <li>1. At 5 in the main flow, if the system is not owned by another player, the use case extends '<b>Buy square</b>'.</li> <li>2. At 5 in the main flow, if the system is owned by another player the use case extends '<b>Pay Player</b>'.</li> <li>3. At 5 in the main flow, if square is a 'chance' square action is taken depending on the card. Player balance is updated, and new balance displayed to the player. The use case extends '<b>Develop Square</b>'.</li> <li>4. At 5 in the main flow, if the player lands on 'blank square' then the use case extends '<b>Develop Square</b>'.</li> <li>5. At 5 in the main flow, if the player already owns the system, then they are prompted if they would like to develop the system. Use case extends '<b>Develop square</b>'.</li> </ol>
Post-conditions	Either the next player 'takes turn' or use case extends another use case from the alternative flows.

Figure 3: Use Case description for 'Take Turn'.

For the 'Buy Square' use case to be initiated the player must have landed on an unowned system square. As part of the main flow, the player successfully purchases the system with their balance being appropriately updated. There are two alternative flows which result in the system square being auctioned, whereby the player refuses the purchase or has insufficient funds for the transaction to be completed. The use case has been executed successfully if the system has been purchased or auctioned and the gameplay proceeds to the 'Develop Square' use case.

Use Case	Buy Square
Actors	Player
Pre-Condition	The player must have rolled the virtual dice and landed on an unowned System square.
Main Flow	<ol style="list-style-type: none"> <li>1. The system provides the System type of the square and states that it is currently unowned. It states the price of the square and the player's current balance, with a prompt asking if they would like to buy the square.</li> <li>2. The player enters 'y' to confirm the purchase of the System square.</li> <li>3. The system assigns the square to the player and deducts the price from the player's balance.</li> <li>4. The system then states that the purchase was successful and provides the updated balance of the player.</li> </ol>
Alternative Flows	<ol style="list-style-type: none"> <li>1. At 2. In the main flow the player chooses not to purchase the element they have landed on. The element is put up for auction between the other players. Each of the other players is asked to submit a bid with the element being assigned to the highest bidder and their bid being deducted from their current balance. If more than one player submits the highest bid, the square is assigned to the player who placed it first. In the case that there are no bids placed that are greater than or equal to the initial cost, the square remains unowned.</li> <li>2. At 3. In the main flow the cost to purchase the system is greater than the current balance of the player. The system notifies the player that they have insufficient funds to complete the purchase. The element is put up for auction between the other players. Each of the other players is asked to submit a bid with the element being assigned to the highest bidder and their bid being deducted from their current balance. If more than one player submits the highest bid, the square is assigned to the player who placed it first. In the case that there are no bids placed that are greater than or equal to the initial cost, the square remains unowned.</li> </ol>
Post-Condition	Extends 'Develop Square'

Figure 4: Use Case description for 'Buy Square'.

The 'Pay Player' use case as shown above extends 'Take Turn' whereby the player has landed on another player's system square. For the main flow, the player pays a fine to the owner of the square with the balances of both players updated accordingly. As an alternative flow, the player avoids the fine if they have insufficient funds. The successful execution of the use case has been achieved if the fine is either paid or avoided and the 'Develop Square' use case has been extended.

Use Case	Pay Player
Actors	Player x (Primary) Player y (Secondary)
Pre-conditions	The use case extends ' <b>Take Turn</b> ' for player x. Player x has landed on player y's square.
Main Flow	<ol style="list-style-type: none"> <li>1. System gets the cost of the fine.</li> <li>2. System checks if player x has enough funds to pay player y.</li> <li>3. If player x has sufficient funds, then the player pays player y and the balances of both players are updated.</li> </ol>
Alternative Flow	1. At 2 in the main flow player x has insufficient funds to pay player y. Player x does <b>not</b> have to pay player y. System displays the balances of players x & y.
Post-Condition	Extends 'Develop Square'.

Figure 5: Use Case description for 'Pay Player'.

The 'Develop Square' use case extends 'Take Turn' when a player lands on a system square that they currently have ownership of and at the conclusion of every turn. As part of the main flow for the use case, the player chooses to complete a minor development where the development level for the square is increased by 1. For the alternative flows, the player can opt to skip the development in which case the system loops through the System squares that they own and gives them the option to upgrade. In the case that the System square has a development level of  $\geq 3$ , the player can opt to complete a major development, setting the development level to 6. Once a System square has reached the development level 6 it cannot be upgraded any further. If the player has insufficient funds to complete a development, the upgrade is not completed. After the system has cycled through all of the System squares that a player owns (if any) with the opportunity to upgrade, then their turn ends, and the gameplay passes to the next player.

Use Case	Develop Square
Actors	Player
Pre-Condition	The player must have rolled the dice and landed on a square.



Main Flow	<ol style="list-style-type: none"> <li>1. The player has landed on a System square that they have ownership of and has a development level of less than 3.</li> <li>2. The system displays the development level and the player's balance. The system provides the user with 2 options, they can choose to complete a minor development, or they can skip the development of the System square. The system quotes the System type and the cost of the upgrade.</li> <li>3. The user enters '1' as they have chosen to complete a minor development.</li> <li>4. The system confirms the completion of the development and the development level for the square is increased by 1. The system quotes the name of the element that has been developed and the updated balance for the player.</li> <li>5. The system then cycles through the remaining squares that the player has ownership of and returns to 2. in the main flow. If they own no other squares, the player's turn ends.</li> </ol>
Alternative Flow	<ol style="list-style-type: none"> <li>1. At 3. in the main flow the player enters '2' as they have chosen to skip the upgrade of the System square. The system then cycles through the remaining squares that the player has ownership of and returns to 2. in the main flow. If they own no other squares, the player's turn ends.</li> <li>2. At 1. in the main flow the player has landed on a System square that has a development level that is greater than or equal to 3. The system displays the development level and the player's balance. The system provides the user with options, they can choose to complete a minor development, a major development or they can skip the development of the System square. The system quotes the System type and the cost of the upgrade. If the user enters '2' The system confirms the completion of the development and the development level for the square is increased to 6. The system then cycles through the remaining squares that the player has ownership of and returns to 2. in the main flow. If they own no other squares, the player's turn ends.</li> <li>3. The player has landed on a 'Chance', 'Blank', or unowned 'System' square the system cycles through the squares that they have ownership of at the conclusion of their turn and returns to 2. in the main flow. If they own no other squares, the player's turn ends. The player is unable to develop the square that they have purchased on the same turn.</li> </ol>
Post-Condition	If an element has reached the development level 6, it cannot be developed any further. Next player takes their turn.

Figure 6: Use Case description for 'Develop Square'.

Use Case	End Game
Actors	Player
Pre-Condition	Each of the teams have been created and the game is underway.
Main Flow	<ol style="list-style-type: none"> <li>1. After each round of turns the system asks the user if they would like to keep playing.</li> <li>2. The user enters 'n' and the system processes it to signal the end of the game.</li> <li>3. The system ends the game.</li> <li>4. The state of the game is summarised with the players placed in order by total value (total resources spent on developments + current balance). The balance and number of squares owned for each player is also presented.</li> </ol>
Alternative Flows	<ol style="list-style-type: none"> <li>1. All squares have been developed fully to development level 6. The system then ends the game, and a summary is produced with the players placed in order by total value (total resources spent on developments + current balance). The balance and number of squares owned for each player is also presented. A development history thread is also outputted by the system to show when each of the Systems were purchased and developed.</li> </ol>
Post-Condition	<p>The game has ended.</p> <p>Each user can see which player was winning in terms of total worth and an overview of the gameplay is provided by the development history.</p>

Figure 7 : Use Case description for 'End Game'.

As a pre-condition for the 'End Game' use case the game must already be underway. For the main flow, a player opts to end the game while it is ongoing. The scoreboard is presented by the system with the players placed in order of their total value. The alternative flow for the use case is when all of the system squares have been fully upgraded. In this case, a history of the developments throughout the game is displayed in addition to the scoreboard signalling the game's conclusion.

The layout of the 'virtual' board is displayed below to provide a visual representation of the game. The board design is not implemented as a GUI with the system instead using text as means of communication with the user. This demonstrates where each of the squares are positioned in relation to each other and the attributes associated with each of them. There are two systems which consist of two squares each and two systems made of three squares. Each of the system squares have costs associated with the custodianship and minor/major developments. The board also includes three chance squares and two 'Deep Space' squares which act as blank squares with no functionality. The 'Space Systems HQ' square allows the

players to collect resources each time they pass. As the game is played using a text-based user interface, the system provides phrases to communicate with the user as opposed to a board presentation to display the gameplay. Each of the squares are added to an array list in the order shown in the board design during the game setup. The dice rolls then determine which position in the array list the player has reached, and the system responds with the details of the appropriate square.




<div>Space Systems HQ</div> <div>Collect x resources as you pass</div>	<div>Research</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Research</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Chance</div> <div></div>	<div>Ground Systems</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Ground Systems</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Deep Space</div>
	<div>Chance</div> <div></div>					<div>Lunar Exploration</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>
<div>Deep Space</div>	<div>Satellites/Support</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Satellites/Support</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Satellites/Support</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Chance</div> <div></div>	<div>Lunar Exploration</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>	<div>Lunar Exploration</div> <div>To take charge : X resources</div> <div>Development Cost : y resources</div> <div>Major Development Cost : z resources</div>

Figure 8 : Board Design for ‘ArtemisLite’.

### Realisation (Main Author : JF)

#### Game Setup/Register Player

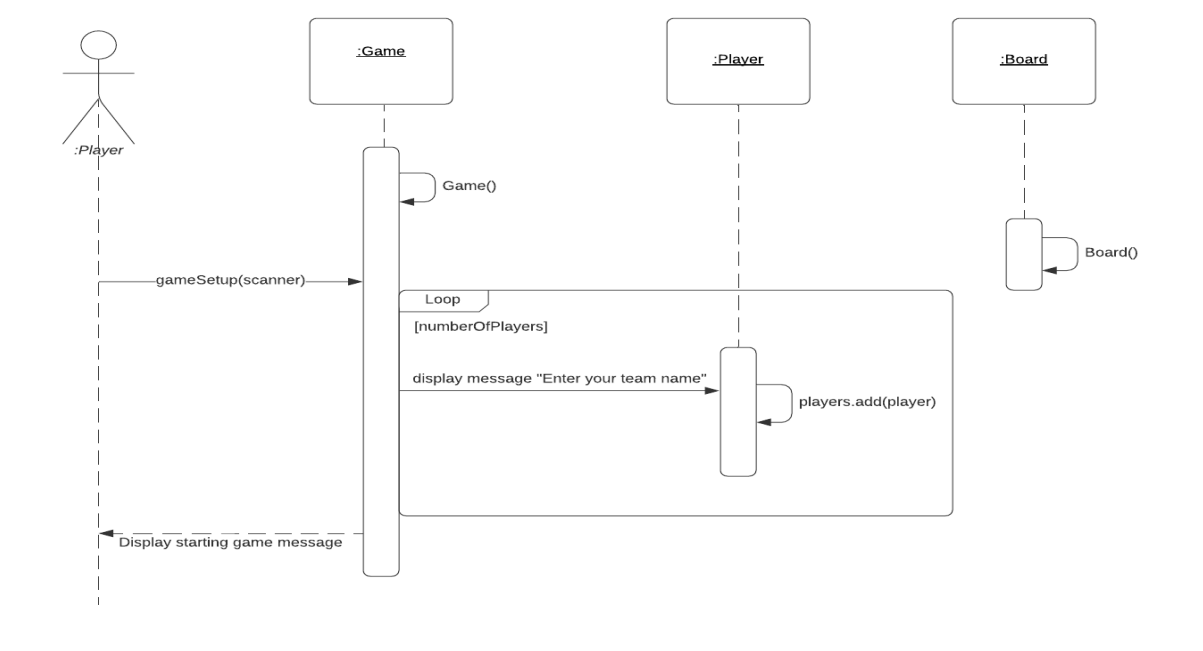


Figure 9 : Sequence Diagram for ‘Game Setup/Register Player’.

The sequence diagram for 'Game Setup/Register Player' demonstrates how the 'gameSetup()' method is called requiring the user to input the number of players. A loop then ensues asking each player to enter their team's name and they are added to the 'players' array list accordingly. The system then displays a message to the user to signal the start of the game.

### Take Turn (Please zoom in to ~ 200% apologies)

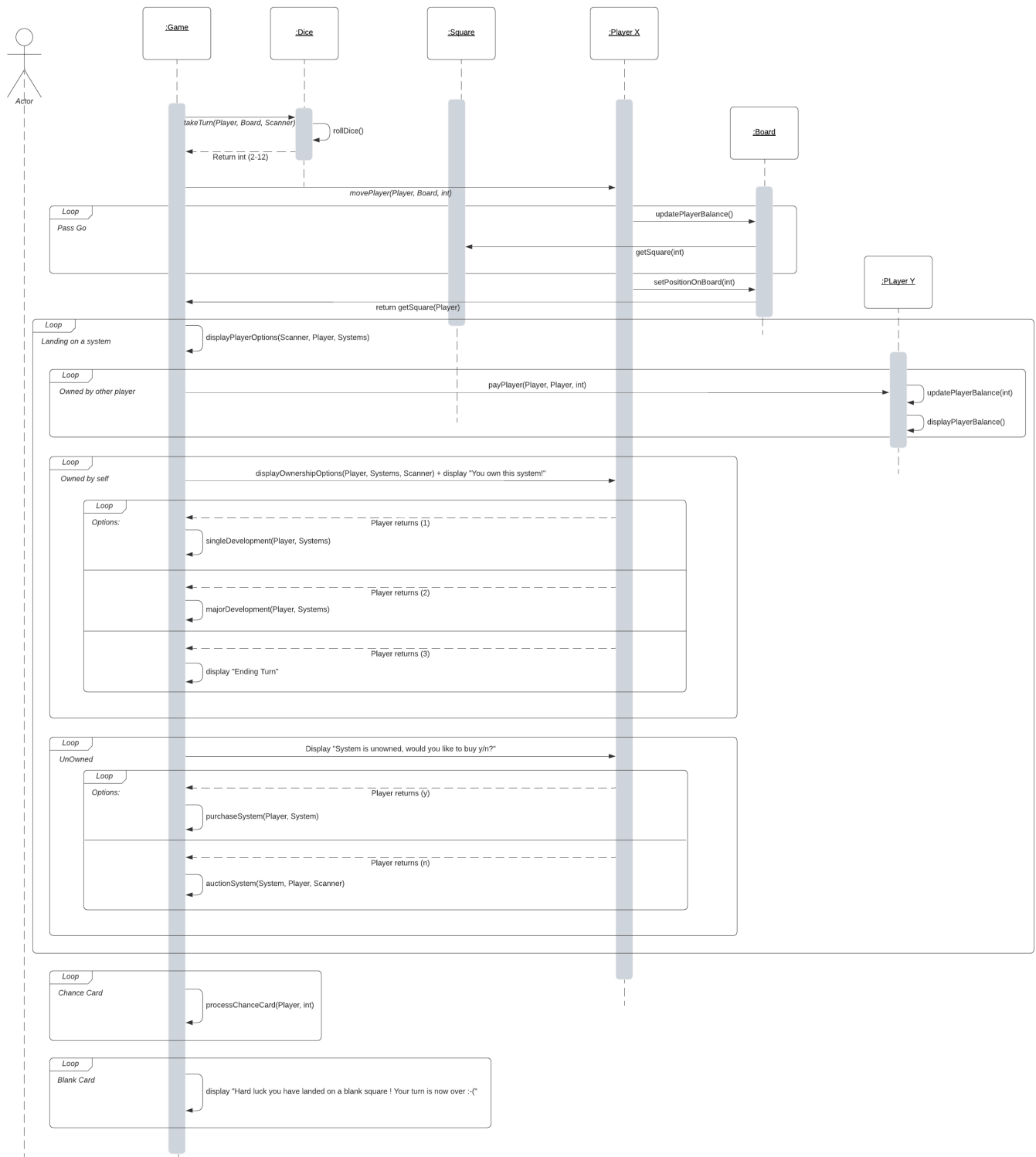


Figure 10 : Sequence Diagram for 'Take Turn'.

The 'Take Turn' sequence diagram displays the 'takeTurn()' method call which includes 'rollDice()'. The 'movePlayer()' method is then called, resulting in four alternative flows depending on the type of square that the player has landed on. If a player has landed on a system square there are three associated alternative flows. Where the system square is already owned by another player, the 'Pay Player' use case is extended. Alternatively, if the player already owns the system square, the 'Develop Square' use case is extended, giving the player the option of a minor/major development. The 'Buy Square' use case is extended if the system square is unowned, resulting in the purchase or auction of the system.

## Chance square

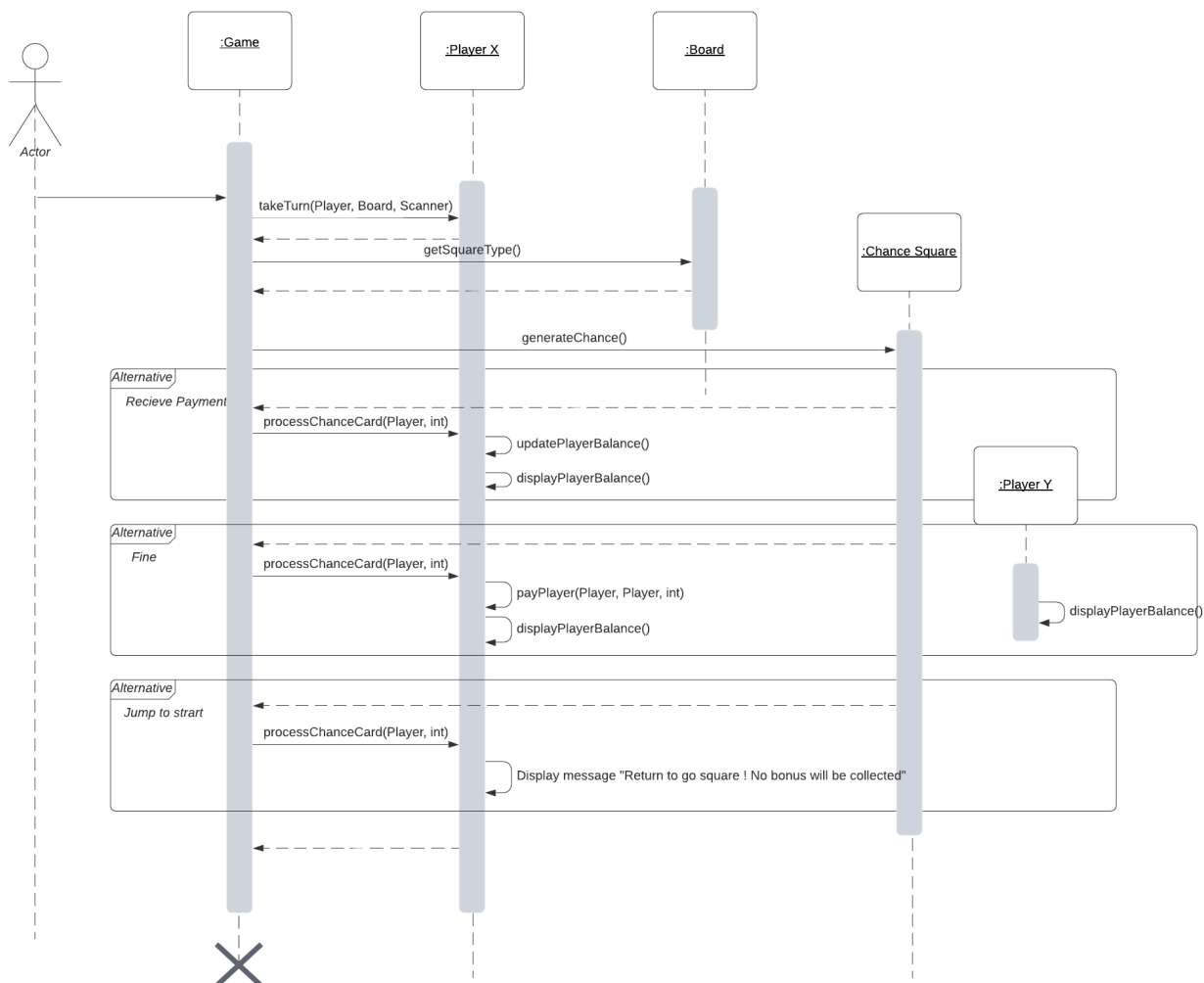


Figure 11 : Sequence diagram for 'Chance Square'.

The 'Chance Square' is an alternative flow for the 'Take Turn' use case. When the 'getSquareType()' method returns 'CHANCE', the 'generateChance()' method is called for which there are three alternative flows dependent on the randomly generated return. The flows include the player receiving a fine, a payment or transport to the 'Start Square' (Space Systems HQ). In each of these cases the 'processChanceCard()' method is called with the randomly generated integer as a parameter. The appropriate methods are called in each case and the 'Develop Square' use case is extended.

## Buy Square

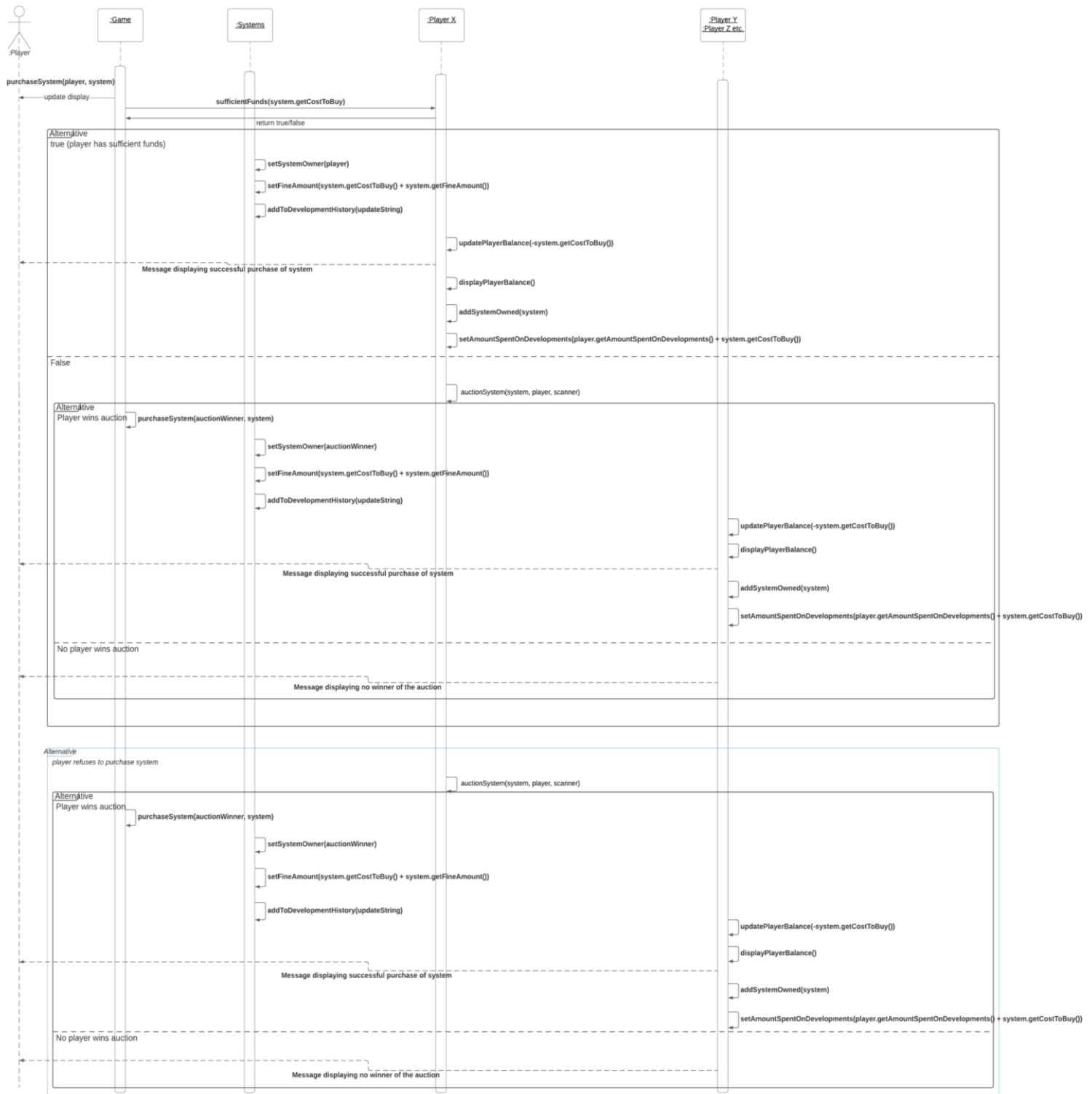


Figure 12 : Sequence Diagram for 'Buy Square'.

The 'Buy Square' sequence diagram shows the 'purchaseSystem()' method being called followed by 'sufficientFunds()' which returns true or false. If the return is true, the player takes

custody of the square with the 'setSystemOwner()' method call. The 'updatePlayerBalance()' method adjusts the player's balance according to the cost of the square. A message is displayed to the user to confirm the purchase of the system square.

If the return of the 'sufficientFunds()' method is false, the system square is auctioned using 'auctionSystem()' with the appropriate square as a parameter. If there is a winner of the auction the 'purchaseSystem()' method is called with the auction winner as a parameter and their balance is adjusted according to the bid amount using 'updatePlayerBalance()'. A message is displayed to the user to confirm the purchase of the system. In the case that there is no winner of the auction, there is an appropriate message displayed.

When a player opts to avoid the purchase of a system square, the 'auctionSystem()' method is called and the other players have the opportunity to make the purchase. The 'Develop Square' use case is extended from 'Buy Square'.

## Pay Player

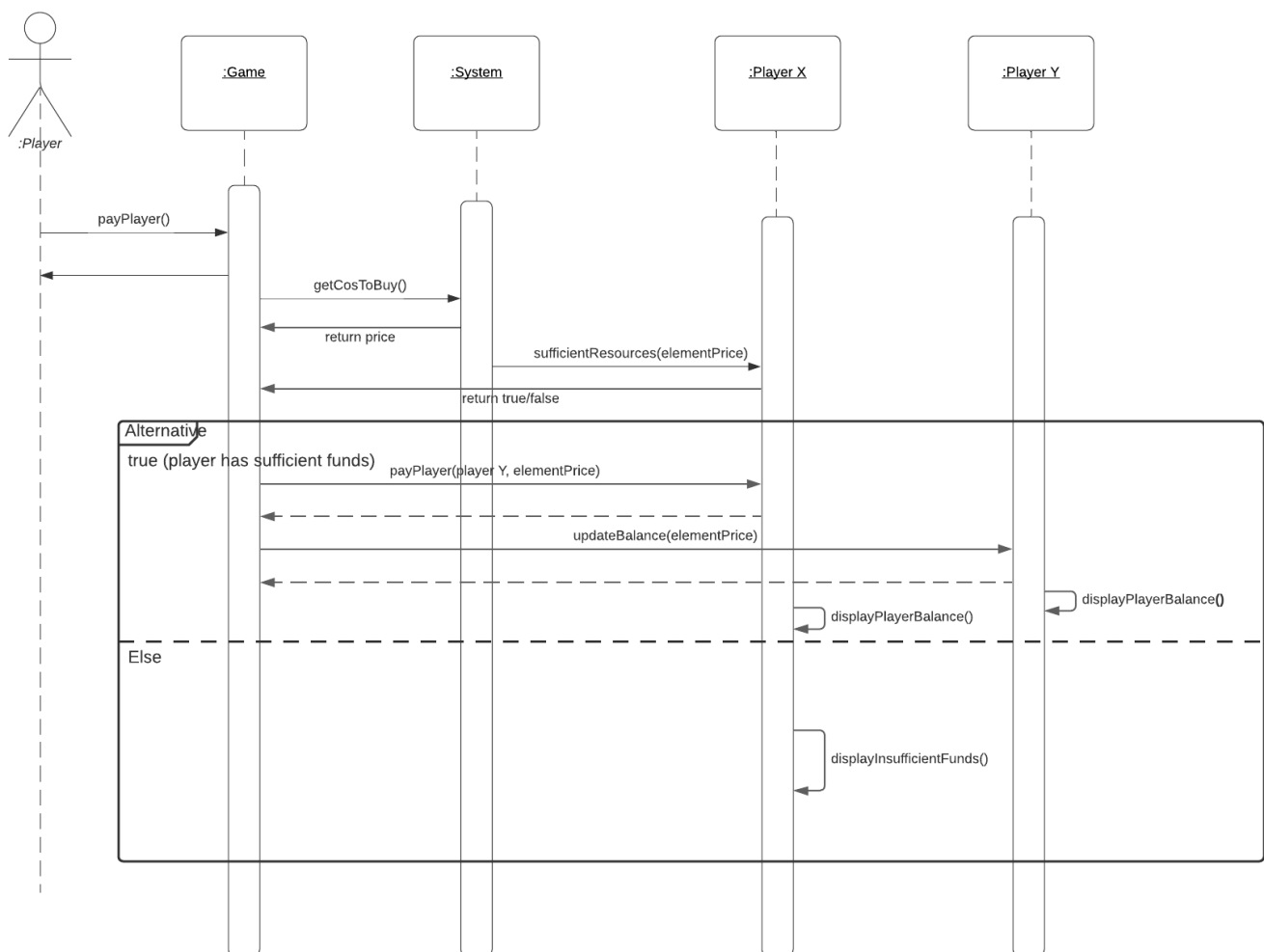


Figure 13 : Sequence Diagram for 'Pay Player'.

'Pay Player' extends the 'Take Turn' use case where a player has landed on a system square that another player has ownership of. The 'payPlayer()' method is called and the 'sufficientResources()' method checks that the player has enough resources to pay the fine. If

the return is true, the `payPlayer()` method is called with the square owner as a parameter. The balances of both players are adjusted by the magnitude of the fine and the `'displayPlayerBalance()'` method is called. As an alternative flow, if the player lacks the resources to pay the fine, the `'displayInsufficientFunds()'` method is called and their turn ends.

## End Game

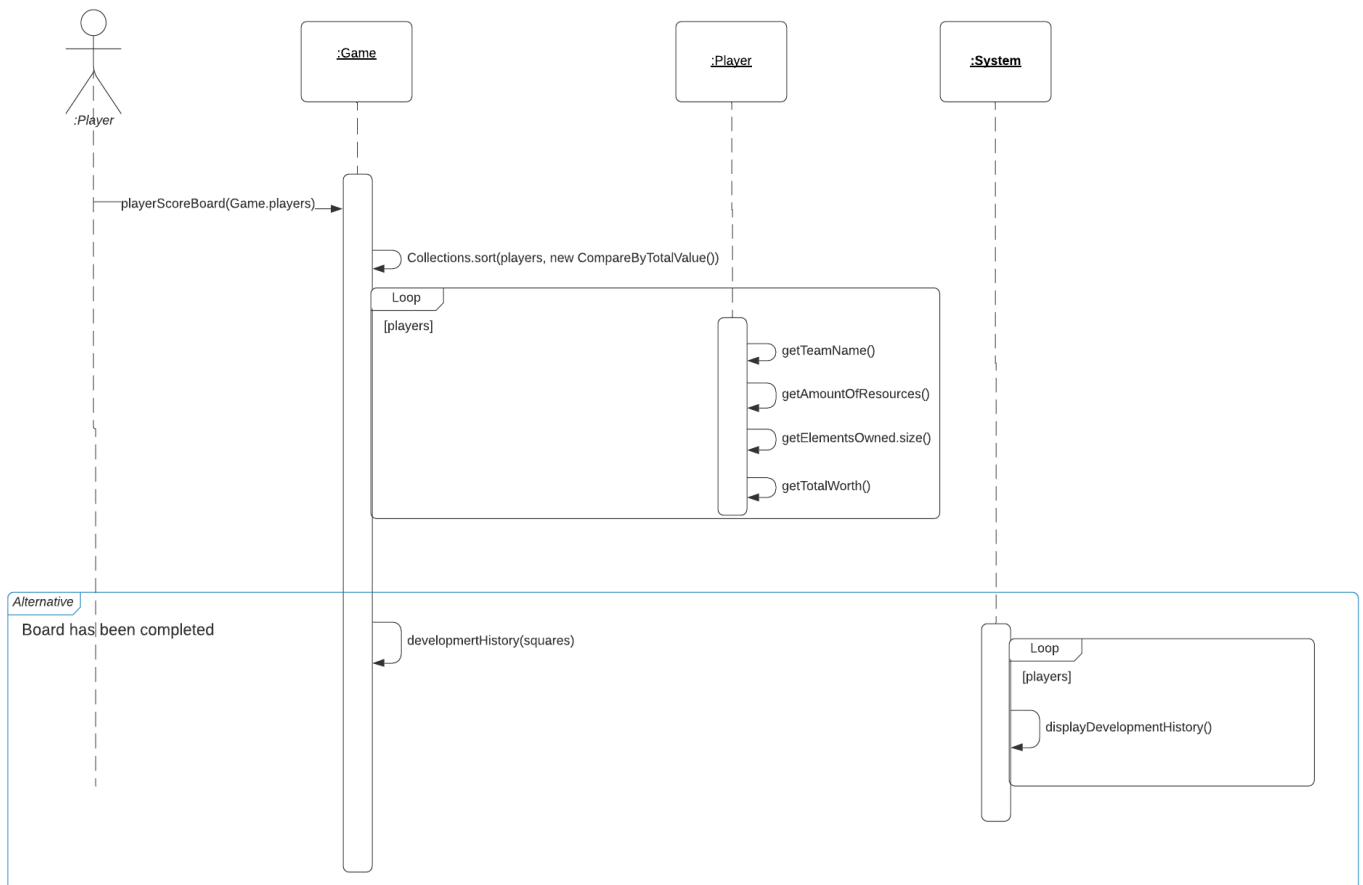


Figure 14 : Sequence Diagram for 'End Game'.

As part of the 'End Game' use case, the `'playerScoreBoard()'` class is called with the `'players'` array list as a parameter. The `'collections.sort()'` method is contained which arranges the players in descending order by total value using the `'CompareByTotalValue'` comparator class. There is a loop of the `'players'` array list with the remaining resources, number of systems owned, and total worth displayed for each player. The `'developmentHistory()'` method is also called in the case that the game has been completed. There is a loop of the `'players'` array list with the `'displayDevelopmentHistory()'` method called. This provides a summary of the purchase and upgrades of the system squares throughout the gameplay.



## Develop Square



Figure 15 : Sequence Diagram for 'Develop Square'.

'Develop Square' extends the 'Take Turn' use case where a player has landed on a system square that they already have custody of. It also extends 'Buy Square' and 'Pay Player' at the end of a turn. The 'systemsOwnedUpgradeOptions()' method is called which contains 'getDevelopmentLevel()'. If the development level is  $< 3$  there are two options displayed by the 'displayOwnershipOptions()' method. If the player enters '1' the 'singleDevelopment()' method is called which will increase the development level of the system by 1 providing they have sufficient funds to complete the transaction. If the user selects '2' the development of the System square is skipped, and the sequence diagram is restarted if the player owns other system squares. After the system has looped through each of the player's System squares the player's turn ends.

Another alternative flow for 'Develop Square' is when the development level is  $\geq 3$ . In this case there are three options provided to the user. If the user selects '1' a single development is applied to the System square. When the user enters '2' a major development is performed, and the development level of the square is set to 6. If the player has insufficient funds, the development will not be completed. In this case when the user selects '3' the development of the System square is skipped, and the sequence diagram is restarted if the player owns other system squares. When a development level of a System square is 6, it can no longer be subjected to either a single or major upgrade.

Players being given the opportunity to upgrade any of the squares they have custodianship over during their turn, satisfies one of the system requirements. Additionally, it was essential to have a minimum of 3 minor developments applied to each System square before the possibility of a major development. The coding structure of the system allowed this functionality to be achieved.

## **Design (Main Author : KC)**

The class diagram shown below provides a visual representation of how the coded aspect of the game has been implemented including the methods and how each of the classes are interlinked. This gives an overview of the entire system and is coherent with the sequences of method calls that have been presented previously within each of the sequence diagrams. When designing the system, there were considerations towards the maintainability to allow any alterations to be efficient.

Creating the Systems class as abstract aids the extensibility of the system as it is inherited by the four types of system class. This allows changes to the Systems class to apply to all of the systems classes without having to manipulate each of them individually. To add a new System type to the game, it would simply require the introduction of a new class that inherits the 'Systems' class and the addition of a related 'Enum' to the 'SystemType' class.

For each of the various system classes, the cost to buy and the cost of minor/major developments have been declared as static variables. This assists with the code useability as these remain unchanged throughout the course of the game. The 'ResearchSystem' squares were attributed as the least expensive, while the two 'Ground\_System' squares were created as the costliest on the board. The system types with 3 squares were assigned the intermediate pricing as set out in the game requirements. The developmentLevel() method is contained within the 'Systems' class as it is applied to every system type square. A minor development increases the development level for the square by 1, while a major development adds 3 to the level. Whenever a square reaches a development level of 6 it has been fully upgraded and cannot be developed any further. The 'isBoardComplete()' method contained within the board class, loops through the square array list to check if each of the system squares have been

fully upgraded. If the return is false, the gameplay continues however a return of true would signal the end of the game.

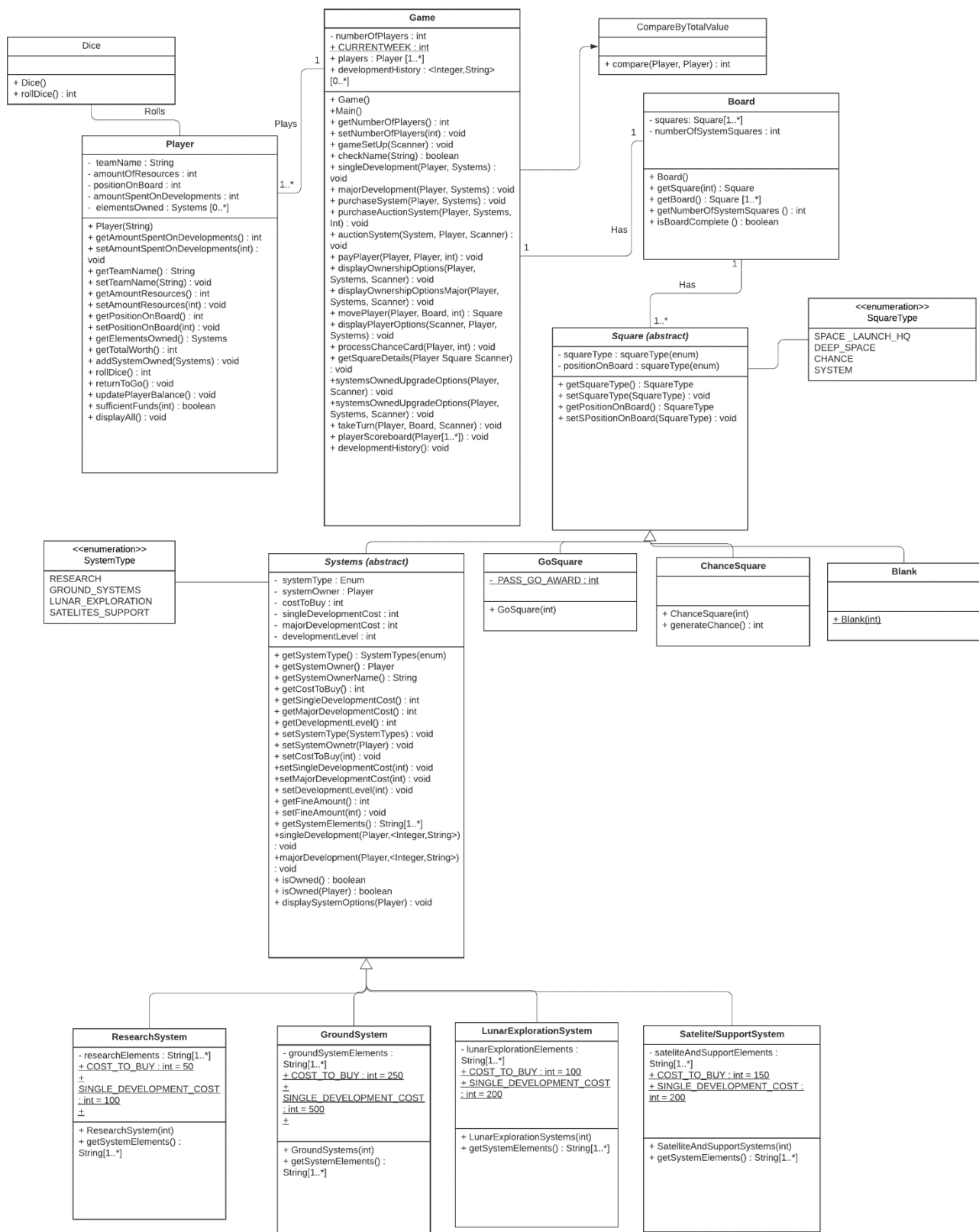


Figure 16 : UML Class Diagram.

At the game's conclusion the 'playerScoreBoard()' class is called to summarise each player's statistics. This method uses the 'CompareTotalValue' comparator class to put the players in order of descending total value. The 'displayDevelopmentHistory()' method employs a thread to output a summary of the transactional events to occur throughout the gameplay. This uses the 'CURRENTWEEK' variable to trace purchase/development actions as it is incremented at the end of each round of turns. The developmentHistory<> tree map is used to store strings regarding developments as values, with the 'CURRENTWEEK' variable matched as the key. As it is a tree map the developments are held in sequence, ascending by current week which is essential to allow the thread to output the events in the order that they occurred.

## Testing (Main Author : PD)

The testing which was carried out took the form of 'Acceptance testing' as well as some J Unit testing (see appendix 1.0). This is a type of testing done by users, customers, or other authorised entities to determine application/software needs and business processes. Acceptance testing is the most important phase of testing as this decides whether the client approves the application/software or not. It is one of the final stages of testing which normally determines if a client will accept the completed product. This is a description on how this testing was carried out during this project. It will methodically run through the different processes within the game and show how it meets the requirements which were set for it.



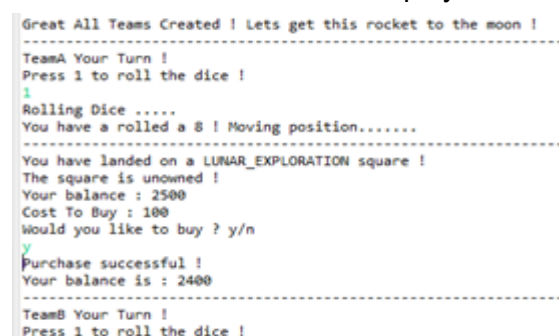
```

Game [Java Application] C:\Program Files\AdoptOpenJDK\jdk-8.0.265.01-hotspot\bin\javaw.exe (21 Apr 2021, 17:42:32)
Welcome to Artemis Lite Game !
-----
Enter how many teams are player ! (2-4)
4
OK! Creating 4 teams!
Player 1, Enter your team name :
TeamA
Player 2, Enter your team name :
TeamB
Player 3, Enter your team name :
TeamC
Player 4, Enter your team name :
TeamD
Great All Teams Created ! Lets get this rocket to the moon !
-----
TeamA Your Turn !
Press 1 to roll the dice !

```

Figure 17 – Creating teams/Assigning team names.

Above shows how at the beginning of the Artemis-Lite game the user interface prompts the user/s to enter the number of players and then to pick a name for the team/organisation.



```

Great All Teams Created ! Lets get this rocket to the moon !
-----
TeamA Your Turn !
Press 1 to roll the dice !
1
Rolling Dice .....
You have a rolled a 8 ! Moving position.....
-----
You have landed on a LUNAR_EXPLORATION square !
The square is unowned !
Your balance : 2500
Cost To Buy : 100
Would you like to buy ? y/n
y
Purchase successful !
Your balance is : 2400
-----
TeamB Your Turn !
Press 1 to roll the dice !

```

Figure 18 – Rolling dice/Purchasing square.

Here notice how 'TeamA' has rolled the dice which outputs a value of the sum of two dice. The player is then moved around the board accordingly. After landing on a square which is unowned the player is prompted if they would like to purchase it while simultaneously being shown their current balance so that they can make an informed decision on whether to purchase the square or not. Once the square is purchased the user's new balance is displayed.

```
-----
TeamB you can now upgrade your other squares if you like ! We will cycle through them now.....
-----
Your balance is : 2300
LUNAR_EXPLORATION
Press 1 to complete a minor development
Press 2 to skip
1
Your balance is : 2100
Minor Development Complete !
Element Developed : Exploration Rover
New Development Level : 1
-----
Week 1 is over ! Check out the standings :

1.Player Name : TeamA
Balance : 2500
Number of Squares Owned : 0
Total Worth : 2500

2.Player Name : TeamB
Balance : 2100
Number of Squares Owned : 2
LUNAR_EXPLORATION on square 8
Development Level : 1
LUNAR_EXPLORATION on square 7
Development Level : 0
Total Worth : 2500
-----
```

Figure 19 – Minor Development.

Figure 20 below illustrates how the user can develop a square which they own even if the user has not landed on that square. This was implemented in order to follow the core requirements set for the project.

```
-----
TeamB - you can now upgrade your other squares if you like ! We will cycle through them now.....
SATELITES_SUPPORT
Press 1 to complete a minor development
Press 2 to complete a major development
Press 3 to skip
2
Your balance is : 1945
Development Complete !
Element Developed : Cargo Units
Development Complete !
Element Developed : Communications Network
Development Complete !
Element Developed : Foundation Surface Habitat
New Development Level : 6
```

Figure 20 – Upgrading squares that the player owns even when they have not landed on them.

```
-----
TeamB - you can now upgrade your other squares if you like ! We will cycle through them now.....
SATELITES_SUPPORT
Press 1 to complete a minor development
Press 2 to complete a major development
Press 3 to skip
2
Your balance is : 1945
Development Complete !
Element Developed : Cargo Units
Development Complete !
Element Developed : Communications Network
Development Complete !
Element Developed : Foundation Surface Habitat
New Development Level : 6
```

Figure 21 – Major development.

Figure 22 shows how if the selected square is of the correct level, then the user is permitted to perform a major development which means that the square has been fully developed. This only will be an option when the system has reached a development level of 3 as is the case in this example.

```

-----
TeamA Your Turn !
Press 1 to roll the dice !
1
Rolling Dice .....
You have a rolled a 3 ! Moving position.....
-----
You have landed on a chance square !
You have been fined for breaching regulations !
Pay TeamB 9$ resources !
-----
TeamA, Updated Balance :
Your balance is : 2485
TeamB, Updated Balance :
Your balance is : 2595
-----

```

Figure 22 – Chance card

Figure 22 is showing how one of the three different chance options (pay player a fine, receive payment, jump to start) is implemented. Once the player lands on chance square one of these options is randomly generated. In this example the player must pay one of the other players. This function is carried out and the balances are updated correctly.

```

-----
TeamB Your Turn !
Press 1 to roll the dice !
1
Rolling Dice .....
You have a rolled a 4 ! Moving position.....
-----
You have passed go !
Collect 200 resources !
-----

```

Figure 23 – Pass go.

Figure 23 shows how if a player passes go, they collect resources for completing a lap of the board.

```

-----
TeamB Your Turn !
Press 1 to roll the dice !
1
Rolling Dice .....
You have a rolled a 8 ! Moving position.....
-----
You have landed on a SATELITES_SUPPORT square !
TeamA owns this square !
You pay them a fine of 100 resources !
TeamB, Updated Balance :
Your balance is : 2300
TeamA, Updated Balance :
Your balance is : 2400
-----

```

Figure 24 – Landing on a square which is owned.

Figure 24 demonstrates a player lands on a square which is already owned by another player then they must pay that player a fine. This fine is dictated by the level of development of that square and the overall importance of that system; some systems are more expensive than others.

```

-----
TeamA Your Turn !
Press 1 to roll the dice !
1
Rolling Dice .....
You have a rolled a 7 ! Moving position.....
-----
You have passed go !
Collect 200 resources !
You have landed on a RESEARCH square !
TeamB owns this square !
You pay them a fine of 550 resources !
You have insufficient funds to pay this player....for the cause of space exploration, you are not required to pay.
-----

```

Figure 25 – Landing on a square which is owned with insufficient funds.

Examining figure 25 notice how when the player lands on the square but has insufficient funds

to pay they are presented with the message 'You have insufficient funds to pay this player...for the cause of space exploration, you are not required to pay.' This was implemented so all players can work together to finish the mission.

```
*****CONGRATULATIONS LANDING SUCCESSFUL ARTEMIS LITE PROJECT COMPLETE !*****
*****
The game is over ! Check out the final standings !
1.Player Name : TeamA
Balance : 2434
Number of Squares Owned : 2
SATELITES_SUPPORT on square 13
Development Level : 6
GROUND_SYSTEMS on square 4
Development Level : 6
Total Worth : 6834

2.Player Name : TeamB
Balance : 519
Number of Squares Owned : 3
SATELITES_SUPPORT on square 12
Development Level : 6
GROUND_SYSTEMS on square 5
Development Level : 6
LUNAR_EXPLORATION on square 9
Development Level : 6
Total Worth : 6219

3.Player Name : TeamD
Balance : 2057
Number of Squares Owned : 3
SATELITES_SUPPORT on square 11
Development Level : 6
LUNAR_EXPLORATION on square 7
Development Level : 6
RESEARCH on square 2
Development Level : 6
Total Worth : 5307

4.Player Name : TeamC
Balance : 345
Number of Squares Owned : 2
LUNAR_EXPLORATION on square 8
Development Level : 6
RESEARCH on square 1
Development Level : 6
Total Worth : 2295
```

Figure 26 – Final leader board.

Figure 26 is an image illustrating how the game ends. The game will automatically end when all the squares have been fully developed. A comparator class will then be used to order the players in terms of 'Total worth.' The player at the top is the winner of the game.

## Appendix I - Testing

Acceptance testing was the main approach to the complete testing of the system as is detailed in the report. Some JUnit test cases were also carried out in order to test some of the foundation classes (those by which other classes are built upon) within the programme. The images below show which methods these tests were carried out on and how successful they were.

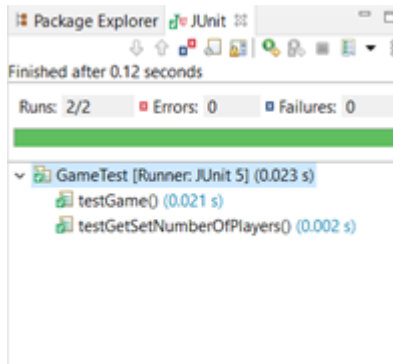


Figure 20 – JUnit test for Game class.

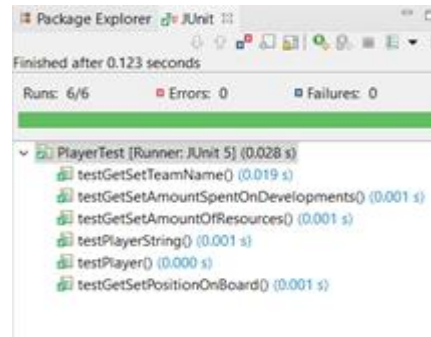


Figure 21 – JUnit test for Player class.

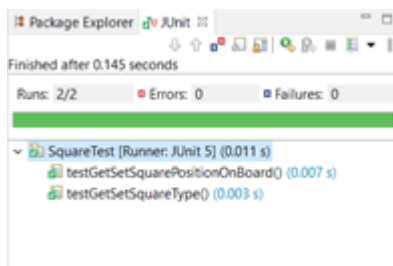


Figure 22 - JUnit test for Square class.

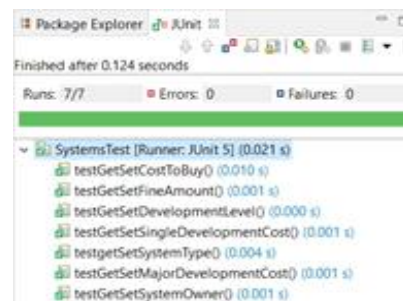


Figure 23 – JUnit test for Systems class.

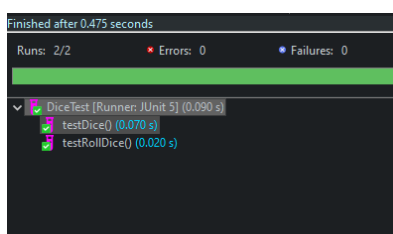


Figure 24 – JUnit test for Dice class.

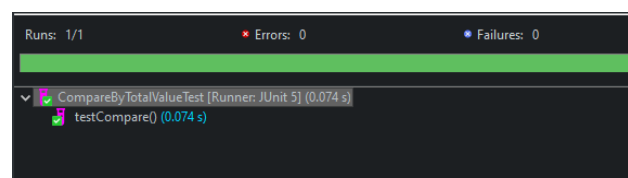


Figure 25 – JUnit test for Dice class

This level of testing was carried out in order to prove that the created system fit the core requirements for the project. Using acceptance testing allows the client to see clearly how the user interface handles itself as the game is being played and easily spot how each the system requirements have been met. The JUnit testing demonstrates how the individual class handles themselves when carrying out instructions from the user/game.

## Appendix – Team Minutes

**Minutes for Group21 Week commencing 22/2 Date of this minute 23/2**



The following team members were present

Name (printed/typed)	Signature
Kyle Carson	-
Jonathan Farrell	-
Pierce Doherty	-

**Task Reporting** (Briefly list the progress for each team member in the last week.\*)

Name & Role (1):

- Kyle Carson – We each created a use case diagram, presented the best of them to the advisors and received feedback.

Name & Role (2):

- Jonathan Farrell – We each created a use case diagram, presented the best of them to the advisors and received feedback.

Name & Role (3):

- Pierce Doherty – We each created a use case diagram, presented the best of them to the advisors and received feedback.

**Actions Planned** (Briefly list the actions required of each team member for the next week.)

Name & Role (1):

- Kyle Carson – After receiving feedback from the use case diagrams we then divided up the use cases and for the following week we will try to come up with some draft versions of the use case descriptions.

Name & Role (2):

- Jonathan Farrell – After receiving feedback from the use case diagrams we then divided up the use cases and for the following week we will try to come up with some draft versions of the use case descriptions.

Name & Role (3):

- Pierce Doherty – After receiving feedback from the use case diagrams we then divided up the use cases and for the following week we will try to come up with some draft versions of the use case descriptions.

**Minutes for Group21 Week commencing 08/03 Date of this minute 09/03**

The following team members were present

Name (printed/typed)	Signature
Kyle Carson	-
Jonathan Farrell	-
Pierce Doherty	-

**Task Reporting** (Briefly list the progress for each team member in the last week.\*)

Name & Role (1):

- Kyle Carson – We each made up some use case descriptions and presented them in the advisory sessions and received some feedback on them. We discussed how the game would be played by reading the artemis lite report.
- 

Name & Role (2):

- Jonathan Farrell – We each made up some use case descriptions and presented them in the advisory sessions and received some feedback on them. We discussed how the game would be played by reading the artemis lite report.

- 
- Name & Role (3):
- Pierce Doherty - We each made up some use case descriptions and presented them in the advisory sessions and received some feedback on them. We discussed how the game would be played by reading the artemis lite report.

**Actions Planned** (Briefly list the actions required of each team member for the next week.)

- Name & Role (1):
- Kyle Carson – After receiving feedback on the use case descriptions we plan to divide up the use cases and try to build some rudimentary sequence diagrams.

- Name & Role (2):
- Jonathan Farrell – After receiving feedback on the use case descriptions we plan to divide up the use cases and try to build some rudimentary sequence diagrams.

- Name & Role (3):
- Pierce Doherty – After receiving feedback on the use case descriptions we plan to divide up the use cases and try to build some rudimentary sequence diagrams.

## Minutes for Group21 Week commencing 22/03 Date of this minute 23/03

The following team members were present

Name (printed/typed)	Signature
Kyle Carson	-
Jonathan Farrell	-
Pierce Doherty	-

**Task Reporting** (Briefly list the progress for each team member in the last week.\*)

- Name & Role (1):
- Kyle Carson – We received feedback on the sequence diagrams from this weeks session. I made a board to help the group to visualise how the game would be played.

- Name & Role (2):
- Jonathan Farrell – We received feedback on the sequence diagrams from this weeks session.

- Name & Role (3):
- Pierce Doherty – We received feedback on the sequence diagrams from this weeks session.

**Actions Planned** (Briefly list the actions required of each team member for the next week.)

- Name & Role (1):
- Kyle Carson – After receiving feedback on the sequence diagrams we will plan to update them for the following week. I also plan to try and create a first draft of the UML diagram.

- Name & Role (2):

- Jonathan Farrell – After receiving feedback on the sequence diagrams we will plan to update them for the following week.

- 

Name & Role (3):

- Pierce Doherty – After receiving feedback on the sequence diagrams we will plan to update them for the following week.

### Minutes for Group21 Week commencing 05/04 Date of this minute : 06/04

Name (printed/typed)	Signature
Kyle Carson	-
Jonathan Farrell	-
Pierce Doherty	-

Task Reporting (Briefly list the progress for each team member in the last week.\*)

Name & Role (1):

- Kyle Carson – I created the first draft of the UML diagram. We all started to divide up the code. I took up a majority of the code and divided the rest up between the group.

- 

Name & Role (2):

- Jonathan Farrell – I started to add all of the material that we had made so far into the report so that we can all see/add and update. I started on some parts of the code.

- 

Name & Role (3):

- Pierce Doherty – I started to help with the code and updated the report as I went.

- 

Actions Planned (Briefly list the actions required of each team member for the next week.)

Name & Role (1):

- Kyle Carson – Will continue with the code and as a group we will start to build up the report.

- 

- 

Name & Role (2):

- Jonathan Farrell – Will continue with the code and as a group we will start to build up the report.

- 

- 

Name & Role (3):

- Pierce Doherty – Will continue with the code and as a group we will start to build up the report.

### Minutes for Group21 Week commencing 12/04 Date of this minute 13/04

Name (printed/typed)	Signature
Kyle Carson	-
Jonathan Farrell	-
Pierce Doherty	-

### Task Reporting (Briefly list the progress for each team member in the last week.\*)

Name & Role (1):

- Kyle Carson – As a group we started to enter the final stages of the report and the code.

Name & Role (2):

- Jonathan Farrell – As a group we started to enter the final stages of the report and the code.

Name & Role (3):

- Pierce Doherty – As a group we started to enter the final stages of the report and the code.

### Actions Planned (Briefly list the actions required of each team member for the next week.)

Name & Role (1):

- Kyle Carson – Will start to move on to the final stages of the project before submission.

Name & Role (2):

- Jonathan Farrell – Will start to move on to the final stages of the project before submission.

Name & Role (3):

- Pierce Doherty – Will start to move on to the final stages of the project before submission.

### Minutes for Group21 Week commencing 19/04 Date of this minute 20/04

The following team members were present

Name (printed/typed)	Signature
Kyle Carson	-
Jonathan Farrell	-
Pierce Doherty	-

### Task Reporting (Briefly list the progress for each team member in the last week.\*)

Name & Role (1):

- Kyle Carson – I took responsibility for much of the code in the final days to make sure that that it work as it was supposed to. I also updated any diagrams that I created throughout the project.

- 

Name & Role (2):

- Jonathan Farrell – I took responsibility for much of the report in the final days and used this time to also update the sequence diagrams that I completed.

Name & Role (3):

- Pierce Doherty – I took responsibility for some of the report and the testing. I also used this time update my diagrams that I was responsible for throughout the report.

### Actions Planned (Briefly list the actions required of each team member for the next week.)

Name & Role (1):

- Kyle Carson – Submission.

Name & Role (2):

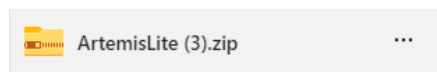
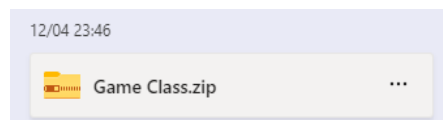
- Jonathan Farrell – Submission.

Name & Role (3):

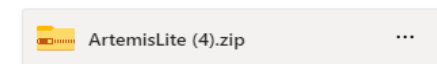
- Pierce Doherty – Submission.

## Appendix III – Code Collaboration

Due to a group member's withdrawal from module and the fact that two of our team were housemates we decided to collaborate on the code in person, working on one device to work faster within the time constraints for the initial implementation of the code. Further on from this we then conducted a process of acceptance testing as mentioned in the report and identified and bugs within the software system. We rectified these bugs as we moved along and updated each other on the alterations to the code via 'Microsoft Teams' chat and posting the code via zip files as shown below.

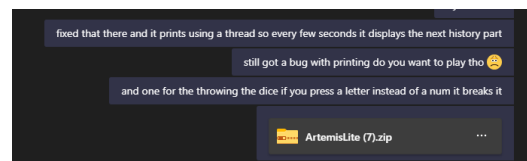
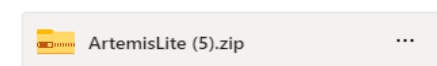


Kyle Carson 14/04 12:36



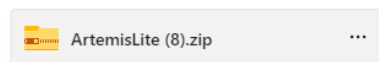
Kyle Carson 14/04 13:14

Fixed some bugs, updated code :



Kyle Carson Yesterday 13:17

- 1)Fixed bug whereby if user inputted a string instead of number for roll dice, an error would occur
- 2)Fixed buy where would you like to player again is duplicated
- 3)Added threads after each printout statement to make it more game like
- 4)Added purchaseSystemAuction method whereby the user purchases the system for the amount they bid (was previously purchasing for the original price of system)



Kyle Carson Yesterday 16:35

Changes

- 1)New method displaysystemowned in players class
- 2)Changed Square Class - now has a parameter called position on board
- 3) Every square constructor - go square, research, blnak etc now has the parameter position on board in its constructor
- 4)All systems classes no longer have the major development cost static variable - this instead is calculated automatically in the systems class
- 5)Methods added in game class ( cant remember them all so please go through them)

