CSC7053 SOFTWARE ENGINEERING 2023
GROUP 17
PROJECT REPORT

Evaluation	Group Number: 17			
Name	Contribution to team-working and motivation[1]	Contribution to documented analysis, design and testing [1,2]	Contribution to working system code[1,2]	**Peer Score** (Range 85 – 115)
Hanbo Li	5	5	5	100
Roberto Lo Duca	5	5	5	100
Sam Millar	5	5	5	100
Dan O'Sullivan	5	5	5	100
Andrew Scott	5	5	5	100

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

[2]This value should consider contributions in the round – direct contributions to required deliverables, and contributions that have made the deliverables possible.

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
Hanbo Li	06/04/2023	I agree to the terms of the declaration
Roberto Lo Duca	06/04/2023	I agree to the terms of the declaration
Sam Millar	06/04/2023	I agree to the terms of the declaration
Dan O'Sullivan	06/04/2023	I agree to the terms of the declaration
Andrew Scott	06/04/2023	I agree to the terms of the declaration

Personal statement of:	Roberto Lo Duca 40386172
The following were my most significant contributions to the project (100 words or less):	

- First Story of the game concept
- Coursework commentary in Requirements elicitation, use case commentary and merging of group use case diagrams by add game maker actor, listed requirements, activity diagram, gameplay and new board expansion
- Purchase area methods, sequence diagram and use case description (+ pass Sunrise desc)
- Board, event, card and failure classes of the code + additional methods in other classes

Personal statement of:	*Hanbo Li*
The following were my most significant contributions to the project (100 words or less):	

- Rent payment design and conduct
- Dutch auction system design and conduct
- Take part in User case design
- Visualized map(Not implemented)

Personal statement of:	*Sam Millar*
The following were my most significant contributions to the project (100 words or less):	

- Team organisation and sprint planning
- Prototyping and class design
- GameSystem logic, user experience and narrative coherence
- Customisable CSV file and dynamic file reading for board setup

Personal statement of:	*Dan O'Sullivan*
The following were my most significant contributions to the project (100 words or less):	

- Initial csv file creation
- GameSetup method
- Dice method and class
- Report editing

Personal statement of:	*Andrew Scott*
The following were my most significant contributions to the project (100 words or less):	

- Develop area method and related methods/subclasses.
- Planned and delivered the presentation.
- Developed the first board visualisation concept.
- Came up with the use cases: move, roll dice and view information.

Contents

Introduction: Solaropoly

For this Software Engineering project, we were tasked with designing and building a console-based game from a set of user requirements, loosely based on the board game Monopoly but on the theme of "Save Our Planet". This report sets out our design process, following an initial waterfall model in requirements elicitation and analysis, product description and system design, followed by a Scrum implementation phase with several sprints including component, integration and system testing and design feedback, finishing with a plan for acceptance testing.

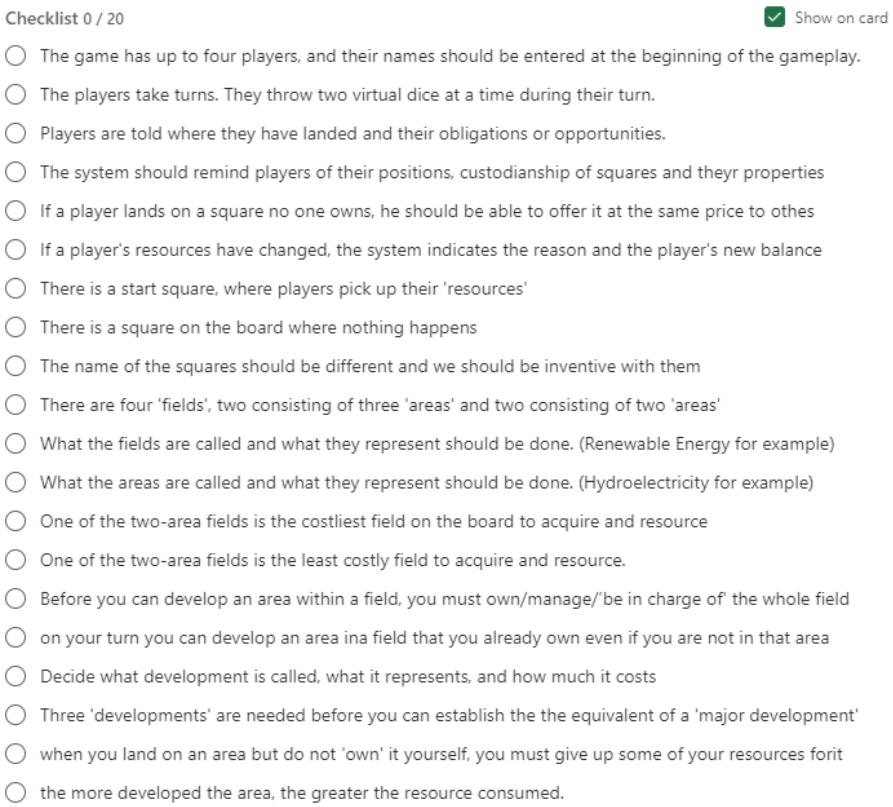
We adhered to Unified Modelling Language v2 design principles in our presentation of a Use Case Diagram, Activity Diagram, Design Class Diagram and Sequence Diagrams. We implemented the system in Java 17 using the Eclipse IDE, all five of us collaborating via GitLab.

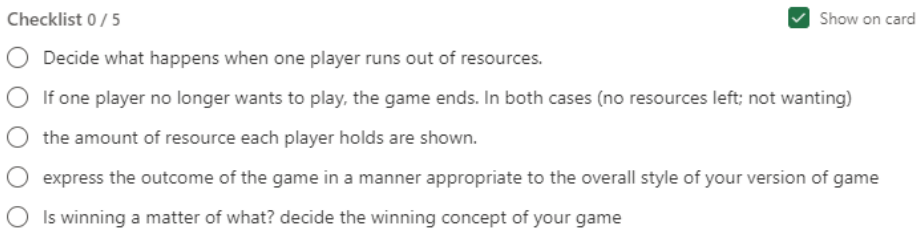
Requirements elicitation & analysis

Identification and shortlisting of candidate use cases

After closely reviewing the requirements, we identified the use cases that were needed to satisfy the core requirements of the game. Some use cases could be added later but were not essential for the initial project Minimum Viable Product.

Based on the project specifications, we created a list of 25 system requirements:





This list serves as a guide for our team as we work towards building the MVP. It includes essential features such as user registration and login, game setup and initialization, turn-based gameplay mechanics and player score tracking etc. In combination with the use case diagrams, this list will be used as a foundation for future improvements, by continuously refining the game and eventually adding new features for the client.

Use cases and use case diagram

Actors that interact with our system:

😊 **Game maker:** The player that starts the game by selecting the number of players.

😊 **Active player:** The player who is currently taking their turn. They will make a series of decisions and interact with other players.

😊 **Competitor:** The player who is waiting their turn. Can be called to action when the active player decides to auction one of his owned properties.

Each actor is part of “All players” which is connected with a “Generalisation” arrow in the use case diagrams below.

We divide our diagram into two nested levels to give a better understanding of the interactions between competitors and the active player:

☐ **Game System:** This box represents the entire system. Outside of the Game System are association lines that connect players that interact with the system.

☐ **Round System:** This box represents the single turn and the various interactions that happen throughout it with both the Active player and Competitor(s).

Everything outside the Round system is considered part of the use cases that don’t represent the turn itself and are usually activated at the beginning or end of the game. Some of these were specifically conceived at the beginning of our development, after the requirement analysis that took place during group meetings.

- **Set number of players:** The whole process of setting the game’s initial status. In our specific case the use case will allow the “Game Maker” to choose how many players are going to play the game and starts it.
- **Register player:** All the players included the “Game Maker”, so all the actors should register and put a unique name on the game registration process. An error message will appear for empty inputs or duplicate names. The Association arrow connects all the players.

- **End game:** The closure system of the game. It is again associated with all players. It shows a message with details for each player, their scores and a message that tells the result of the game that is connected with the game story itself. End game can end whenever the players want to conclude the game or if the number of players is less than 2 (no reason at that point to play alone).

Inside the round system itself, initially with a large set of different use cases, we decided to simplify the visual aspect of our system to avoid confusion and misinterpretations. So, we decided to merge some of the most important aspects and sequence of actions in distinct and essential use cases:

- **Roll dice and move:** Asks the player if they want to start then roll the dice or end the game. It is included in the Move use case because it will be always triggered every turn to allow the user to move. That permits the player to move in the board with a series of actions. Roll dice is the first of the series of actions that will be shown to the player before starting their turn and shown his opportunities during the turn. Roll dice is included in move, so if the player does not trigger this use case, the game will end.
- **Purchase area:** This is the most common use case in the early game. It gives the option to the player to buy initial areas and sets players up to be able to auction or develop their areas. As specified in the requirements, if it is not possible to buy an area because the amount of resources is too low or because the player does not want to, the game gives a series of choices to the player. He could sell the area to other players and interact with them in the system to allow them to accept or reject the offer and the player gets to choose who should be the final owner. It is associated with both the active player and competitors because they need to act during this event. Purchase area, like reward owner, is also an extension of the square 'move' because it is not always triggered, for example if the area is already owned it will trigger the reward owner function instead.
- **Develop area:** This use case is used to allow the player to develop and increase the productivity of their clean energy factories. This is one of the core requirements of the game and allows the mechanics of the game to be more complex. This is asked every turn as is stated in the requirements.
- **Auction area:** This is not a requirement, but we added it as a feature. We have many special features in the structure of our game and one of these is the flexibility to allow the client to add features in the project. We decided to create a use case description for this new feature. Auction area allows the user to acquire other players properties. Like the Purchase area it requires interaction with the competitors.

These 3 + 1 represent the series of actions that the Active player can take during his turn. They represent the core requirements of the game as they allow the game to be playable.

Finally, the user cases we developed explain the mechanics of the systems and their interactions with the player's balance, status and position in the board. These are required to run and develop the game, return clear results and updated status to the player:

- **Move:** This use case allows the player to move through the board. The player position will be updated, and a series of events will trigger based on the act method of each square on the board. This use case will trigger after the dice roll. For example, land in an Area owned by another player will trigger the Pay rent use case.

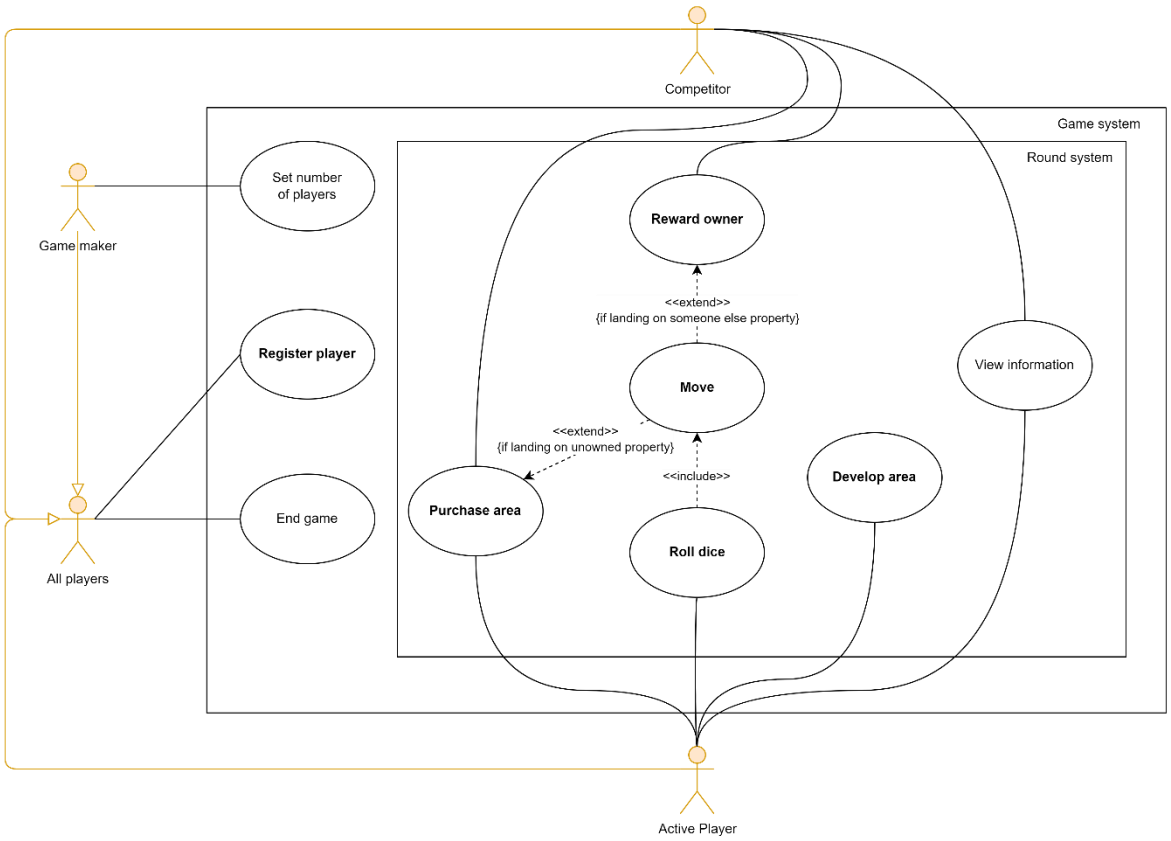
- **Reward owner:** This use case is activated when the move use case brings the active player to an area owned by another player. To satisfy the requirements we decided that the player will lose part of his resources, in this case GETs, whenever the player lands on an area of land owned by a competitor. This use case is directly associated with the competitor and the active player can do nothing more than suffer a loss on his credits. Reward owner is an extension of move because the whole process is not always triggered by the move use case, but only if we land in an owned area. Instead, if the area is not owned the other extension that will trigger as said before is the purchase area.

- **View information:** We need to satisfy the requirement of the user interface experience to make the game playable and enjoyable, hence why the game logic alone is not enough. We decided to add a use case called View information that will trigger each turn to display all the squares, properties, position of the player, balance and status in the game. To do so we elaborated on an easy to grasp interface to allow the user to go through the game with a clear understanding. We also separated each turn of the player with a break line to attract the attention of the next player.

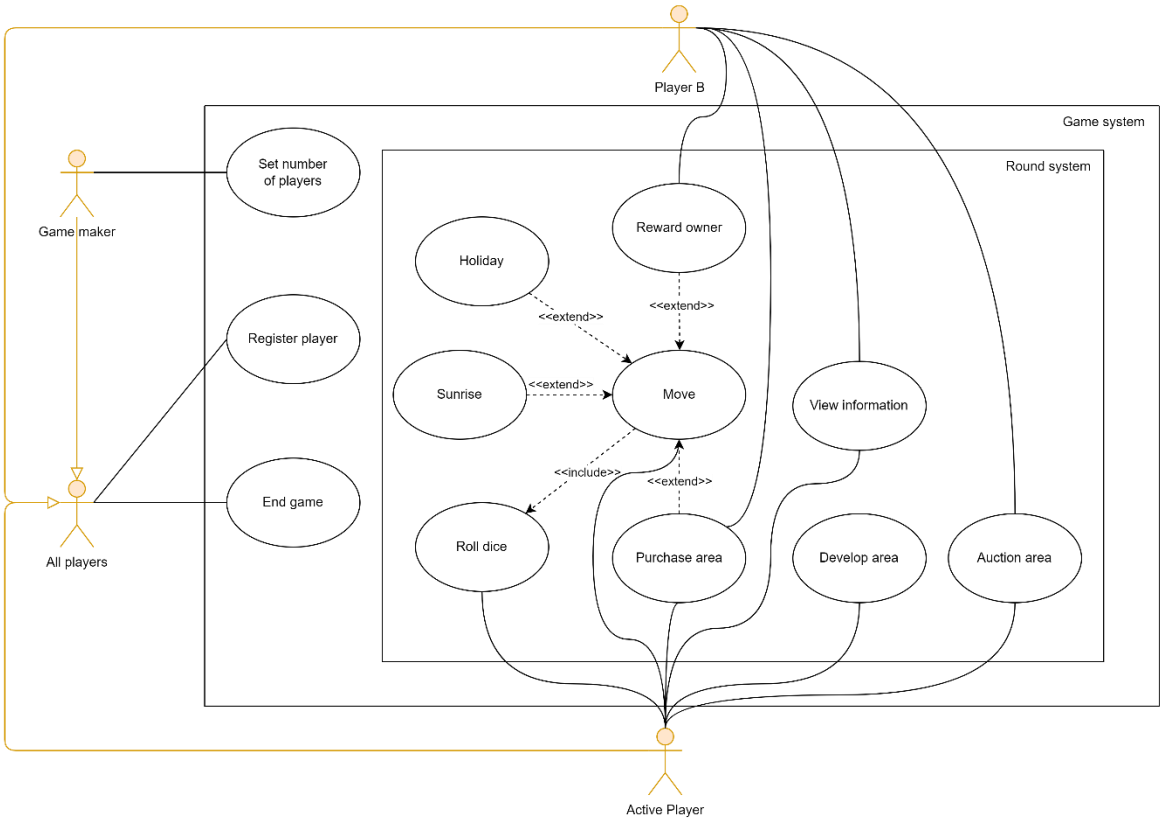
Overall, these are the most important use cases that we selected to build the fundamentals of our version of the game. The Move use case also includes other extensions like other squares that are in the requirements, like the Sunrise and the Holiday square:

The Sunrise acts like the starting point on the board and gives the player resources when it is passed, as per requirements. We called it Sunrise because is a mix of meanings: Sunrise happens at the beginning of a new day (new cycle on the board) and the sun brings new energy for solar panels (the resource we get when we pass it is GET). Meanwhile, the Holiday square represents the required square that does nothing.

These two squares are simple, and we decided to keep the diagram simple in keeping with the main logic of the working product. This is also part of the requirements, and a use case description is added for them. The use cases that permit the game to be runnable are shown in diagram 1, while diagram 2 adds landing on squares “Sunrise” and “Holiday” and our expansion the Auction/Trade feature.



Use case diagram 1 – required and lumped use cases



Use case diagram 2 – expanded and split use cases

Use case descriptions

These descriptions clarify the sequences of actions and mechanics of each use case on the diagram. Writing these was an extremely useful step, allowing us to think clearly about how the system should behave, prior to designing and building the components.

Name	**Set number of players**
Objective	To ensure the system caters for the correct number of players
Actors	Organiser (any prospective player)
Precondition	Game has started running; welcome screen displayed.
Main Flow	1. The system asks any player to enter the number of players who wish to play the game. 2. Someone enters 2, 3 or 4.
Alternative Flow	At 2 – someone enters any other sequence. 3. The system advises this is invalid input and asks again for the number of players. Re-join Main Flow at 1.
Post-condition	Number of players stored. System ready to register players by name.

Name	**Register player**
Objective	To register a player for the game.
Actors	Prospective player
Precondition	Number of players has been set.
Main Flow	1. The system requests name input from the next prospective player. 2. The prospective player enters their name. 3. The system records the new player and either loops from 1 for the next player or starts a new game (if all registered).
Alternative Flow if all-whitespace input	At 2 – Someone sends blank input (Enter only). 3. The system advises names must have at least one non-whitespace character. Restart Main Flow at 1.
Alternative Flow if a player enters the same name as a player already registered	At 2 – Someone inputs the same name as a previously registered player. 3. The system advises that players should have unique names. Restart Main Flow at 1.
Post-condition	New player registered. System ready to register next player or start game if all players registered.

Name	**Take turn**
Objective	To roll the dice, move on the board, and receive information about any consequences.
Actors	Active player
Precondition	Normal gameplay phase. It's the player's turn.
Main Flow	1. The system displays the player's balance.

	2. It asks them to press Enter if they would like to take their turn or type "Quit" and Enter to end the game. 3. The player presses Enter. 4. The system rolls the virtual dice, changes the player's location on the board, and informs them of the roll and landing and any consequences (change in balance) or required action, e.g. • Landed on Sunrise • Landed on Holiday • Landed on an unowned Area • Landed on an owned Area
Alternative Flow if invalid input	At 3 – the player inputs text other than "quit". 4. The system advises this is invalid input. Restart Main Flow at 1.
Post-condition if landed on Sunrise or Holiday	System updated with new balance and position for player. Turn proceeds to development phase if player has fields to develop, else ends and play passes to next player.
Post-condition if landed on owned square and bankrupt	System updated with new balance and position for player. Play ends and players' final status is shown.
Post-condition if landed on an unowned square	System updated with new balance and position for player. System offers to sell the property to the player.

Name	**Leave game**
Objective	To leave the game, returning all assets to the UN.
Actors	Active player
Precondition	Play is in the roll and move phase of a turn.
Main Flow	1. The system displays the player's balance. 2. It asks them to press Enter if they would like to take their turn or type "Quit" and Enter to end the game. 3. The player types "quit" and presses Enter. 4. The system displays the assets they had, then removes them from the turn cycle and removes their ownership from all properties.
Alternative Flow	At 3 – the player inputs text other than "quit". 4. The system advises this is invalid input. Restart Main Flow at 1.
Post-condition	Play proceeds to the next player if there is more than one remaining; if there is only one player remaining the game end is triggered.

Name	**Develop field**
Objective	To develop property on areas in fields/groups owned by a player.
Actors	Registered player.
Precondition	The player has rolled and moved on the board. The player owns one or more whole fields/groups which are not already fully developed. The player has sufficient balance to fund one or more developments.

Main Flow	1. The system advises the player which fields/groups they may add a development to and the cost of doing so and asks them to press Enter (if only one option), the index of the field/group (if multiple), or any other character if they do not wish to develop a field/group. 2. The player inputs Enter/1/2/3/4. 3. The system advises which areas within the field may be developed and asks them to press Enter (if only one option), the index of the area (if multiple) , or any other character if they do not wish to develop a square. 4. The player inputs Enter/1/2/3. 5. Steps 3-4 recur until the player indicates they do not wish to develop the field further, they run out of funds, or development is maxed out. 6. Steps 1-5 recur for all applicable fields/groups unless the player indicates they do not wish to develop any further.
Alternative Flow if accidental negative input at 2	At 2 – the player accidentally provides input other than Enter/1/2/3/4. 3. The system asks the player to confirm they wish to stop developing. 4. The player indicates they wish to keep developing. Restart Main Flow at 1.
Alternative Flow if accidental negative input at 4	At 4 – the player accidentally provides input other than Enter/1/2/3. 5. The system asks the player to confirm they wish to stop developing the current field. 6. The player indicates they wish to keep developing. Reset Main Flow to 3.
Post-condition	System updated with new developments. Play passes to next player

Name	**View information**
Objective	To display relevant information to player
Actors	Active player
Precondition	None
Main Flow	The player can see their status. This includes their names, resource amount and area(s) owned. The player can also see view information about areas on the board. This includes the names of the areas, the landlords of the areas, the number of developments (inc. if major development) on each area and the price to purchase the area.
Alternative Flow	None
Post-condition	None

Name	**Purchase area**
Objective	To allow a player to purchase a property or trade them at their usual price. These squares are usually properties that can be bought and developed by players to earn a reward at the expense of the other player who land on them.
Actors	• Active player • Competitors (when the square is sold by the active player)
Preconditions	- The game is settled by the game maker (Set number of players) - The player is registered in the game (Register player) - The player chooses to roll the dice and play (Roll dice) - The active player has landed on an Area square (Move)

	- The area is not owned by anyone....
Main Flow 1	1. The game system presents the player with the options to buy the property, sell it at the same price or none of the 2 options. 2. The player decides to invest on the property (buy) and uses his credits to obtain it. 3. The game system transfers ownership of the property to the player.
Alternative Flow 2 - if at point 1.2 the active player decides to not buy the property, or he cannot afford it	1. The game system informs the active player that they do not have enough GETs and offers them the option to sell it to other players (competitors) at the current price or do nothing. Or: the active player himself from the beginning decide to sell even if he has enough credits. skip the error message if was chosen by the active player to sell it. 2. The game system deducts the necessary amount from the competitors balance and allows them to buy the property automatically by asking the option to the competitors that can obtain it. All the allowed players should accept or decline the offer (If no one accepted the offer will be considered rejected from all). 3. The active player chooses between who accepted the offer who should obtain the property. Ship this point if the competitor to choose is only one and go directly to point 4. 4. The game system transfers ownership of the property to the chosen competitor.
Post-condition	The player (active or competitor/s) now owns the property and can develop it to earn a reward on it at the expense of other players who lands on it. The game system updates the game board to reflect the player's ownership of the property and the development status of the property. One of the messages to the active player if the square was offered is: congratulations, your square was sold, of it was bought: Ok, you've bought (name of the square/area).
Alternative Flow 3	If at point 2.1 or 1.1 the active player decides to do nothing, the post-condition 2 will trigger.
Alternative Flow 4	If at point 2.2 none of the competitors has enough credits to accept the property, the post-condition 2 will trigger.
Alternative Flow 5	If at point 2.2 all the competitors reject the offer of the active player, the post-condition 2 will trigger.
Post-condition 2	The game system does not transfer ownership of the property and does not update the property status (do nothing). The player lost the opportunity to buy the property and to receive credits from it. Said to the player: no one wants your square, so it will be offered to the next person who lands on it.

Name	**Pass Sunrise**
Objective	This use case describes the actions that occur when a player passes the "Sunrise" square on the game board.
Actors	Active player
Precondition	- The game is settled by the game maker (Set number of players) - The player is registered in the game (Register player) - The player chooses to roll the dice and play (Roll dice)

	- The player is currently located or passed the "Sunrise" square during their turn
Main Flow	1. If the player lands on or passes the "Sunrise" square, the system will send a message to the player informing him about the payment 2. The system will award the player with the resource 3. The system will announce the player's new balance and update their total amount of game currency.
Alternative Flow	The amount will not change without the preconditions (Pass Sunrise)
Post-condition	The player's balance has been updated with the appropriate amount of game currency. The player's turn continues normally.

Name	**Transfer investment (“rent” payment)**
Objective	Transfer GET to the player in whose square the active player has landed
Actors	Active player
Precondition	The player has rolled and moved onto others land. The player has sufficient balance to pay the rent fee.
Main Flow	1. After roll, system shows the sign that player has already stepped to other’s land 2. System calculates the total fee need to pay by acting player 3. System automatically transfer money from active player to landowner. 4. Go to next player’s round
Expansion: **Alternative Flow 2 if in flow 1.2 player has not enough money to pay**	1. The game system informs the player that they do not have enough funds to pay for it 2. System calculates all mortgaged value of all assets 3. System shows the balance of his account, rent he needs to pay 4. System shows recent asset list of this player 5. System asks player does he or she want to sell major development if the player has it. If player does not have it, jump to 2.8 6. Player shall type in the number of which major development does he want to sell 7. Player receives money from system. If player has enough money to pay, system shows it and go to next round. If player has not enough money to pay but still has hotels, system shows the rest of rent need to pay and jumps to 2.4. If player has not enough money and players does not have other hotels, it jumps to 2.10 8. System shows recent asset list of this player 9. System asks player does he or she want to sell houses if the player has it. If player does not have it, jump to 2.12. 10. Player shall type in the number of which houses does he want to sell 11. Player receives money from system. If player has enough money to pay, system shows it and go to next round. If player has not enough money to pay but still has houses, system shows the rest of rent need to pay and jumps to 2.8. If player has not enough money and players does not have other houses, it jumps to 2.12 12. System shows rent asset list of this player

	13. System asks player does he or she want to sell land if the player has it. If player choose no, it jumps to 2.5 14. Player shall type in the number of which lands does he want to sell 15. System checks whether development built on it 16. If there are major developments built on it, it jumps to 2.4. If there are houses on it, it jumps to 2.8 17. Player receives money from system. If player has enough money to pay, system shows it and go to next round. If player has not enough money to pay but still has major developments, system shows the rest of rent need to pay and jumps to 2.4. If player has not enough money but has houses, system shows the rest of rent need to pay and it jumps to 2.12
Alternative Flow 3 if in flow 2.3 player has not enough money to pay	1. In 2.2, if all mortgaged value of assets is less than what the active player needs to pay, the player leaves the game.

Product description

The following game narrative serves as a user guide.

Overview

Solaropoly is a text-based console game where players take the role of renewable energy startups participating in a United Nations initiative, competing and collaborating to develop ambitious regional solar power projects to replace fossil power and ease the energy crisis impacting living standards across the globe. Players invest in project areas and compete for control of project regions, working out how to arrange the project areas effectively amongst themselves to maximise energy capture in the time available.

Objective

The objective of the game is to generate a target power output of renewable energy (in megawatts, MW) using investments of Green Energy Tokens (GETs), within a limited number of turns. The target is based on the combined output of all players, so players must collaborate effectively by trading project areas and tokens. Additional recognition is made of the player whose investment tips the total power output over the threshold. The initiative will fail if the target output is not reached in time, or if only one player is left in the game.

Game Setup

Each player's solar power startup embarks from the Sunrise square with a seed investment of GETs.

The game board is set up with several regions, each of which comprises a set of two or three project areas that players can buy from the UN using GETs if they roll and land on them. Project areas must be owned as a regional set for them to be upgraded, as they represent complementary components of energy infrastructure. Rural/desert regions have three project areas each: a huge land lease for solar panels, a solar panel production facility, and a grid to transmit the electricity to distant

settlements. Cities, which already have a grid, have only two project areas: a scheme for rooftop solar panel installation and a solar panel factory.

An additional Holiday square on the board has no effects.

The game may also be set up with Event squares (where players draw a random card that may cause them to move, take another turn, or gain or lose GETs) and Failure squares, where players become stuck for several turns.

Gameplay

Players take turns rolling dice to move their company around the game board, buying and selling properties as in Monopoly.

When an area is landed on by a player for the first time, the player may choose to invest GETs to buy the project area, offer the area to other players (they may choose which player if multiple players bud for it), or leave it unsold. Each area has its own cost and maximum solar energy potential, and if they own complete sets of areas, players can choose to invest in expanding and improving their solar farms, photovoltaics factories and grids to maximize their energy output. The more upgrades a player makes, the more electricity the region generates, and the more tokens they attract from competing projects for further investment in purchases and upgrades.

If a player lands on an area owned by another player, GETs are transferred from them to the owner of that area. An owner can attract more GET and generate more energy through vertical integration – buying an entire region, which allows them to upgrade each area up to four times.

Players receive additional investment each time they pass the Sunrise square.

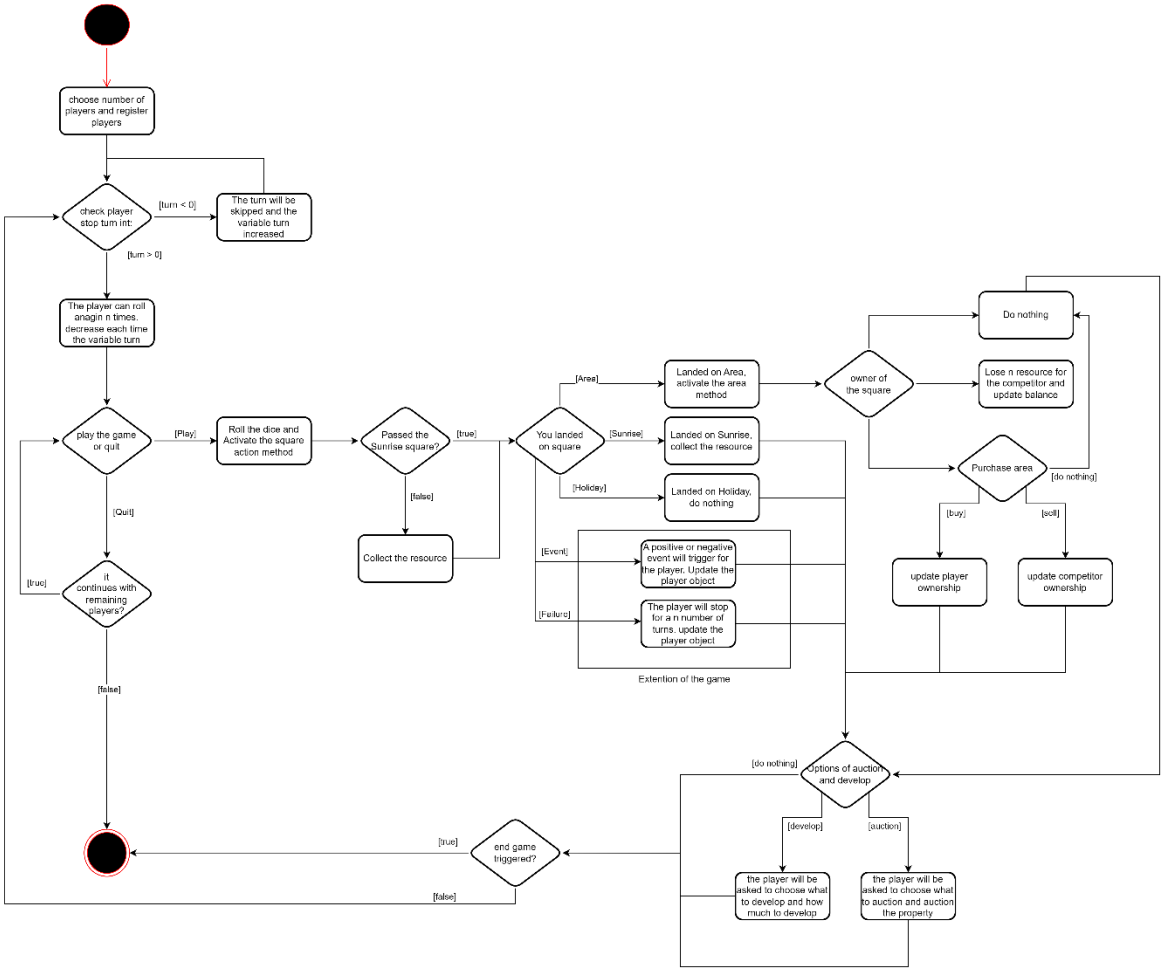
After their move, a player may choose to auction property to the best bidder and/or develop areas in any of their complete regional sets.

The game ends in collective failure if the pre-set number of turns run out or all but one player has left the game, or in success when the combined target power output is reached.

Sales pitch

Solaropoly is a fun and educational board game that promotes optimism, efficiency and collaboration in the face of the climate and energy crises. The game provides an engaging and competitive experience that encourages players to think strategically and make informed investment decisions, not only to maximise their own success but that of everyone.

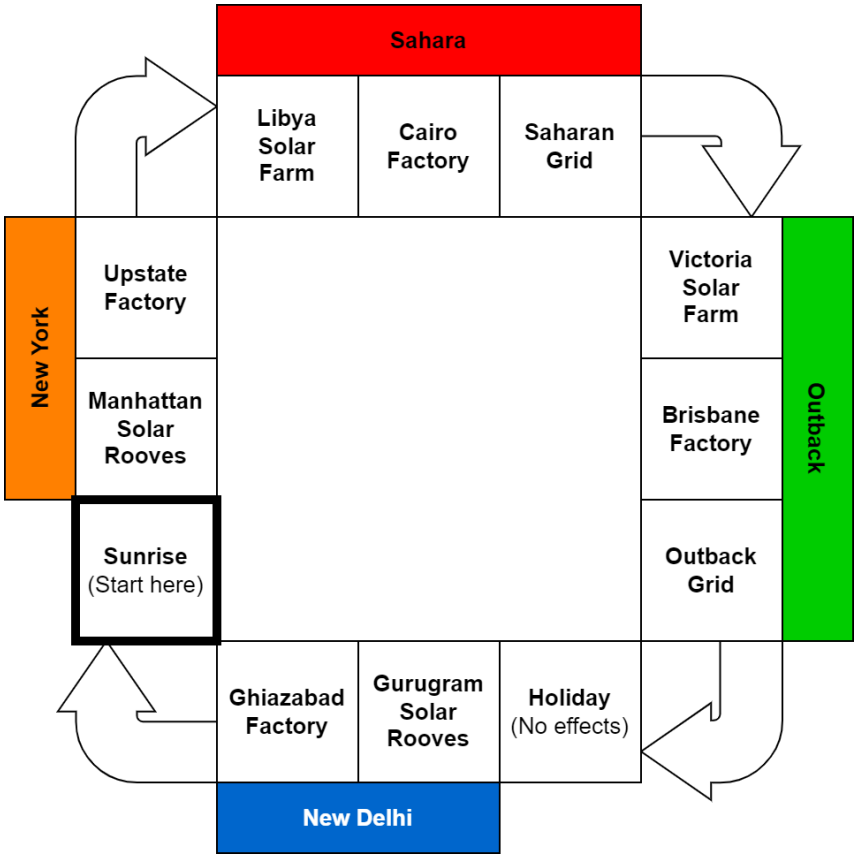
Activity Diagram



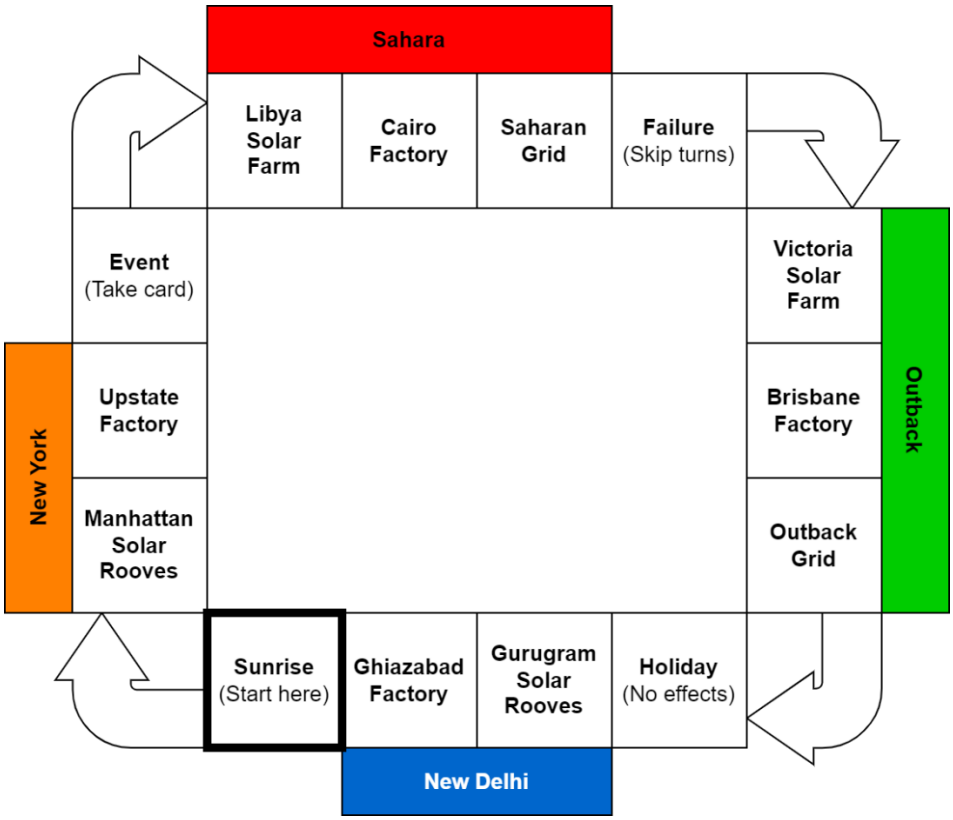
Behavioural Model: Activity Diagram of the game

The activity diagram summarises how the player interacts with the system and the different decision/steps that take place during the game. This version covers the expanded use cases, including the Event/cards and Failure squares and auction system.

Board visualisation



Game board visualisation 1 – adhering strictly to requirements.

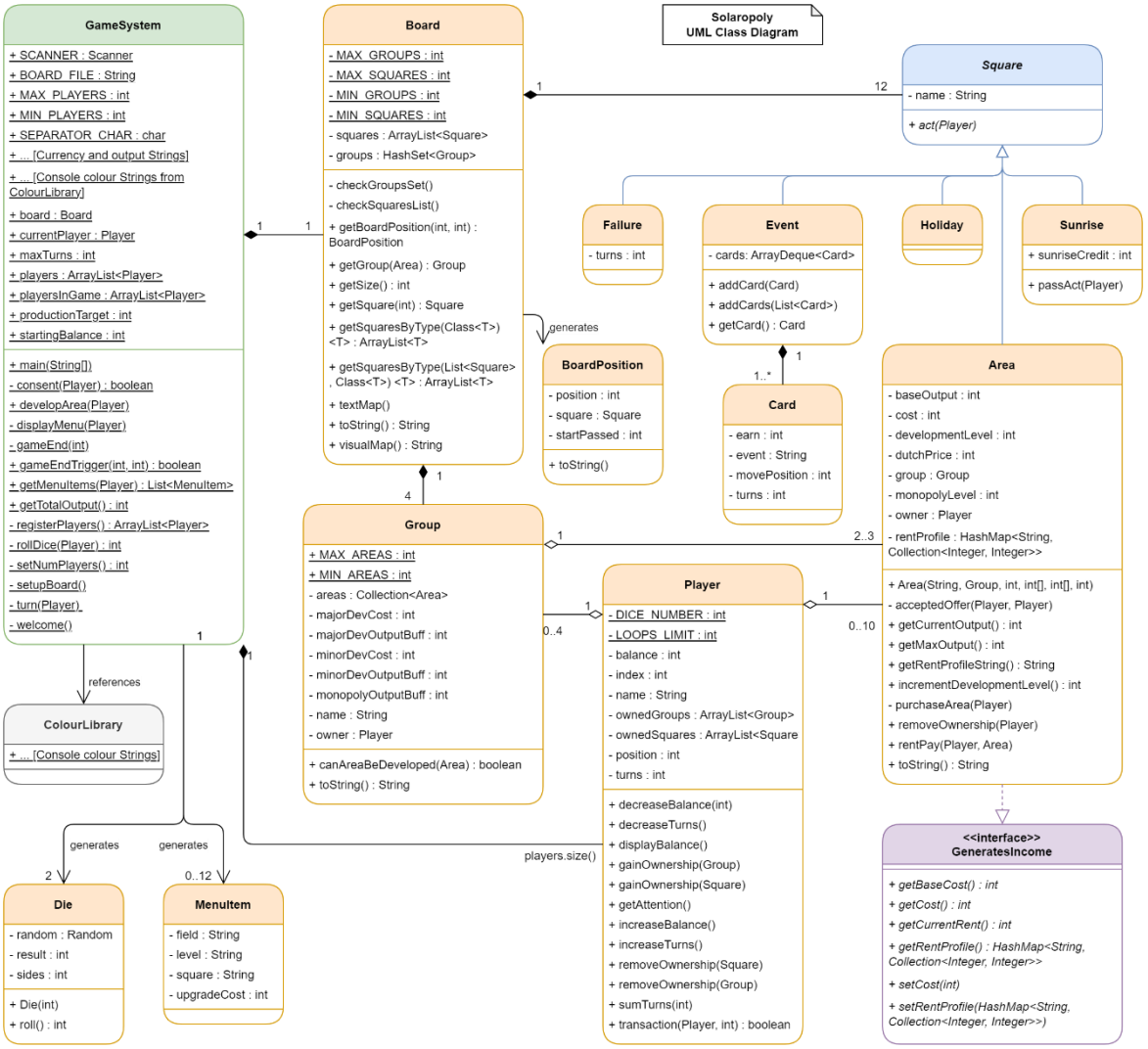


Game board visualisation 2, with optional additional Event and Failure squares

System Design

Having established the sequences of events that would make up each use case in the game, we began developing a class diagram for an Object-Oriented Programming system that would support the game. Based on this we developed Sequence Diagrams for required use cases.

Design Class Diagram



Classes are colour-coded by their function in the system. Green highlights the executable class, orange represents classes that are instanced as objects, blue represents abstract classes, purple represents interfaces, and grey represents library classes with only static fields. UML arrows denoting composition, aggregation, extension, implementation and mere association are used. Composition is understood to mean that if the container class (with the black diamond) was deleted, the references to the contents would be lost. For compactness, standard getters, setters and comparators are not shown, abstract methods are only shown in the superclass/interface and ": void" is omitted.

GameSystem is run by the compiler and manages the overall game flow, including setting up the game board, registering players, managing turns and ensuring the game ends appropriately. It generates pairs of Dice for each roll, and **MenuItems** showing Area information for the development mechanic. It includes as constants a set of console output colours representing a visual design language, which can be flexibly changed by referencing a full library of colour codes held as constants in the **ColourLibrary**. **Die** is instanced twice in GameSystem.rollDice() – dice are created with a validated number of sides (6 for board rolls) and a Random number generator.

A **Board** is created by the GameSystem, populated with the Squares and Groups whose data is provided in the csv file. The class has methods to manage these components as the game progresses, e.g., board.getSquare() which takes a Type argument, returning all Squares of a given extension.

Player represents the player's company, their "piece" on the board, and manages their board position, ownerships and GET balance. The move() method is one of the major methods, taking the dice result and calling the board to generate a **BoardPosition** object which holds a position and associated Square, and checks if the player has passed Sunrise. The move() method also manages trading and development after the player has moved. Other methods include transaction, which handles a safe transfer of GETs from a player to another, and methods to update ownership of areas and groups.

Each **Square** represents an ordered location on the board. This abstract class is defined with a name field and abstract act() method that runs polymorphically when a player lands on it according to one of three required classes (Sunrise, Holiday, Area) and two additional classes (Event, Failure). 12 Squares are held in an ArrayList in the board object as required in the specification – one Sunrise, one Holiday and ten Areas (two groups of two and two groups of three) – but any combination starting with one Sunrise can be set up in the csv file and read in without breaking the logic of the game. The abstraction also allows additional square types to be created – we have set up two examples in the code in the form of Event and Failure.

Sunrise is the Go/Start square, which provides additional investment when a player passes or lands on it (passing requires a separate method which coordinates with the dice roll), while the **Holiday** square has no effects.

Each **Area** represents a facility that can be owned and managed by a player to earn investment (rent) and produce power output. To this end it has an owner, cost (GET), base rent (GET), rent profile (lists of GET values), development level, monopoly level, base output (MW) – plus output buffs for vertical integration, minor development and major development. The class methods include purchasing areas and managing the transfer of ownership.

Two or three Areas are aggregated into a **Group**, which represents the required Field entity. We used Group as a class name because Field is the name of a class in the Java library; ultimately in gameplay we call these Regions. The Group defines common minor and major development costs for the areas within. If a player owns a whole Group, the monopoly buffs are added to the investment and power output, and they may then also increase the development level (to a maximum of 4 which is printed as a major development) attracting additional buffs.

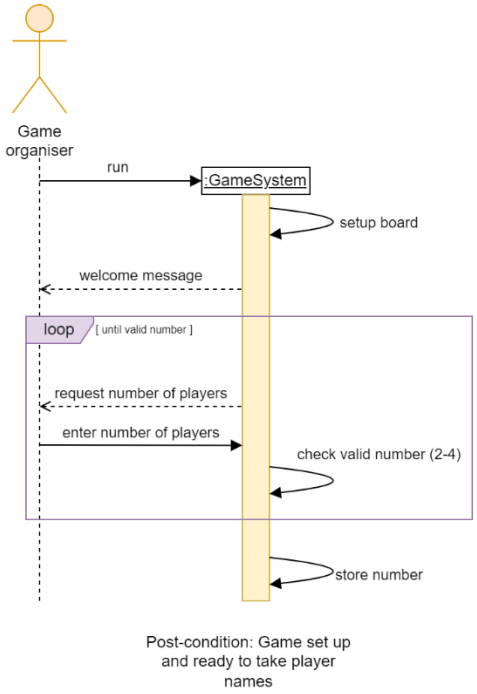
The **Event** square as the name suggests, is a Square extension that holds a series of events (**Cards**) that will trigger after calling the act method of that square. The act method asks the player to take a card and read its content, then the effect will be applied on the player and a series of options could be shown to the player if for example they land in a new square or if they should roll multiple times

in the same turn. To do so we added a class called Cards that stores the different series of events to trigger plus an informative message of the event. After landing on this Square, the player can be affected by a positive or negative event like the reduction of credits and/or affecting his new position by triggering the move method and the act of the new square where they will land. The card can also trigger the block of a series of turns or adding new turns for the player. For example, if a turn is added the player can roll again after the first roll, if it is removed it means that the player will skip his next turn during the game. These parameters are settled in the player class and can be modified by the negative consequences of some of these cards. All the effects of the cards are stored again in the csv file of the game. We can have as many cards as we need and add more of them later. The position of the cards doesn't matter in the csv file as these will be shuffled randomly at the beginning of the game. After shuffling them they are called in a circular order every time one of them is triggered to simulate the effect of putting at the beginning of the deck the last card used. Additionally, the card can decrease, increase the GETs etc. An example of the message of a card could be: "Today is a sunny day!! Your solar panel production increased! You receive n additional GETs" or "there is a problem with a power plant, and they have asked you to go to fix it – go to position n" etc.

The **Failure** class is also an additional square used as an example of extension of the game. This square acts like a failure in one of the power plants and blocks a certain number of turns for the player by setting a variable in the player object. As all the other squares it has its own act method and in this case is simple.

Sequence Diagrams

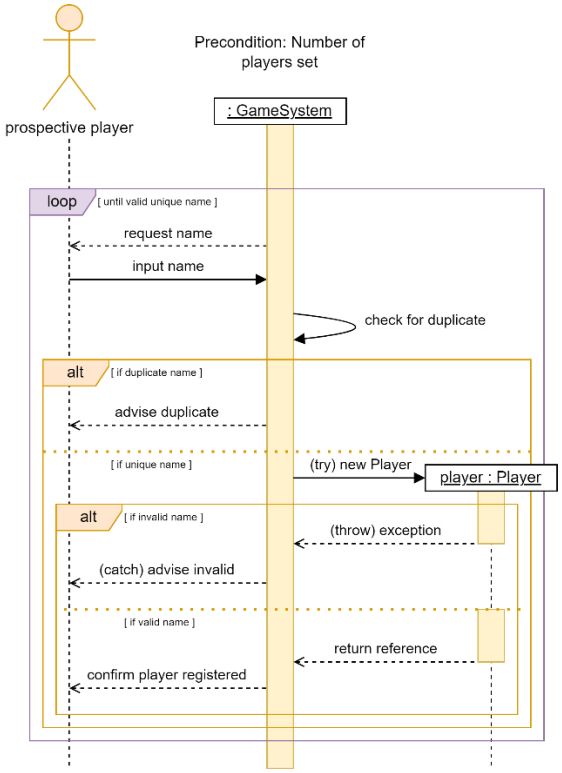
Start game & set number of players



Post-condition: Game set up and ready to take player names

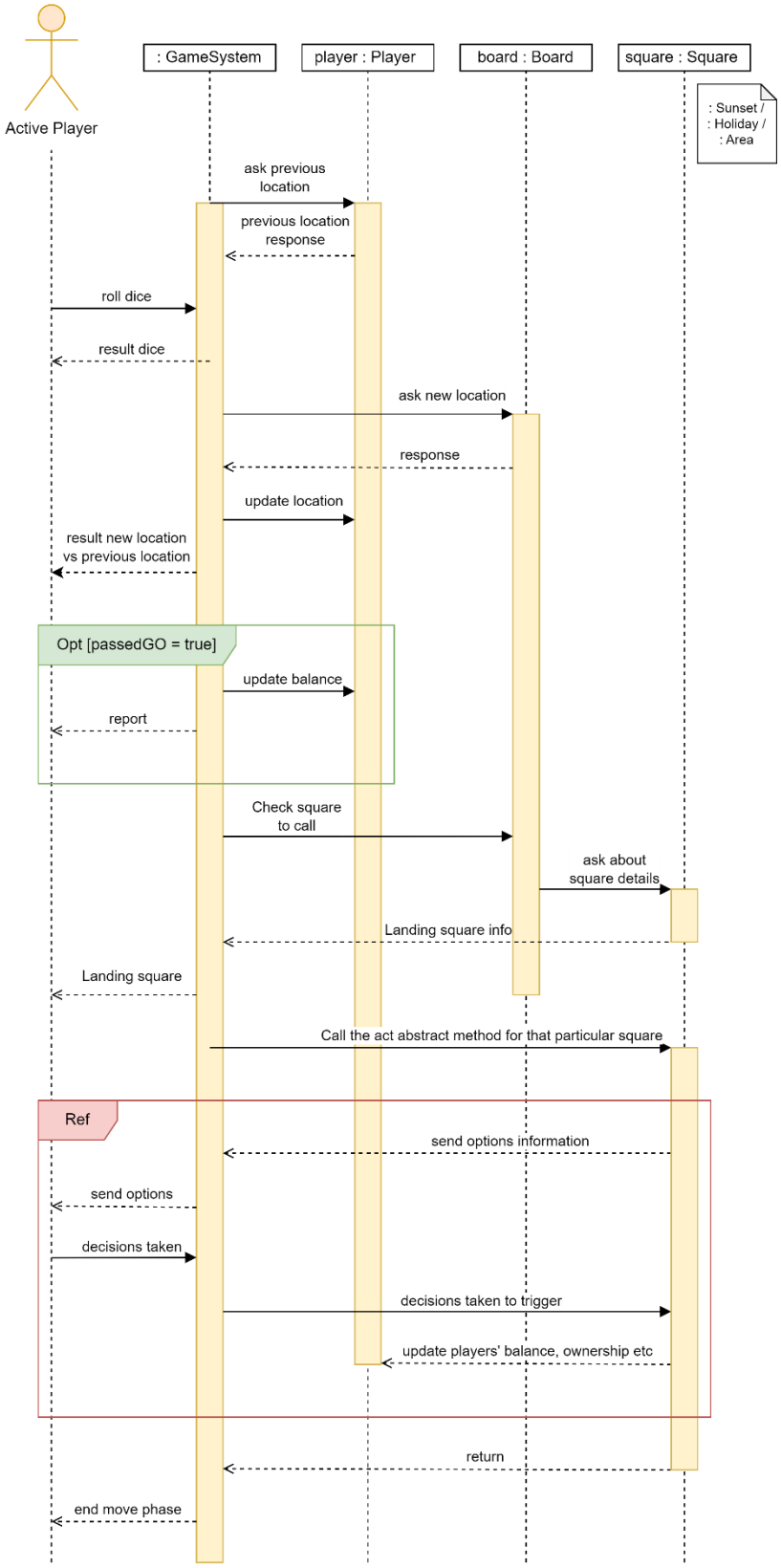
At game start, the board is set up by GameSystem and feeds into a welcome message indicating the starting balance, turn limit, target output and maximum output. Then the number of players is requested (based on the built-in maximum and minimum) and validated in a loop before player registration begins.

Register Player



Postcondition: Game asks next player for their name / asks first player to take their turn

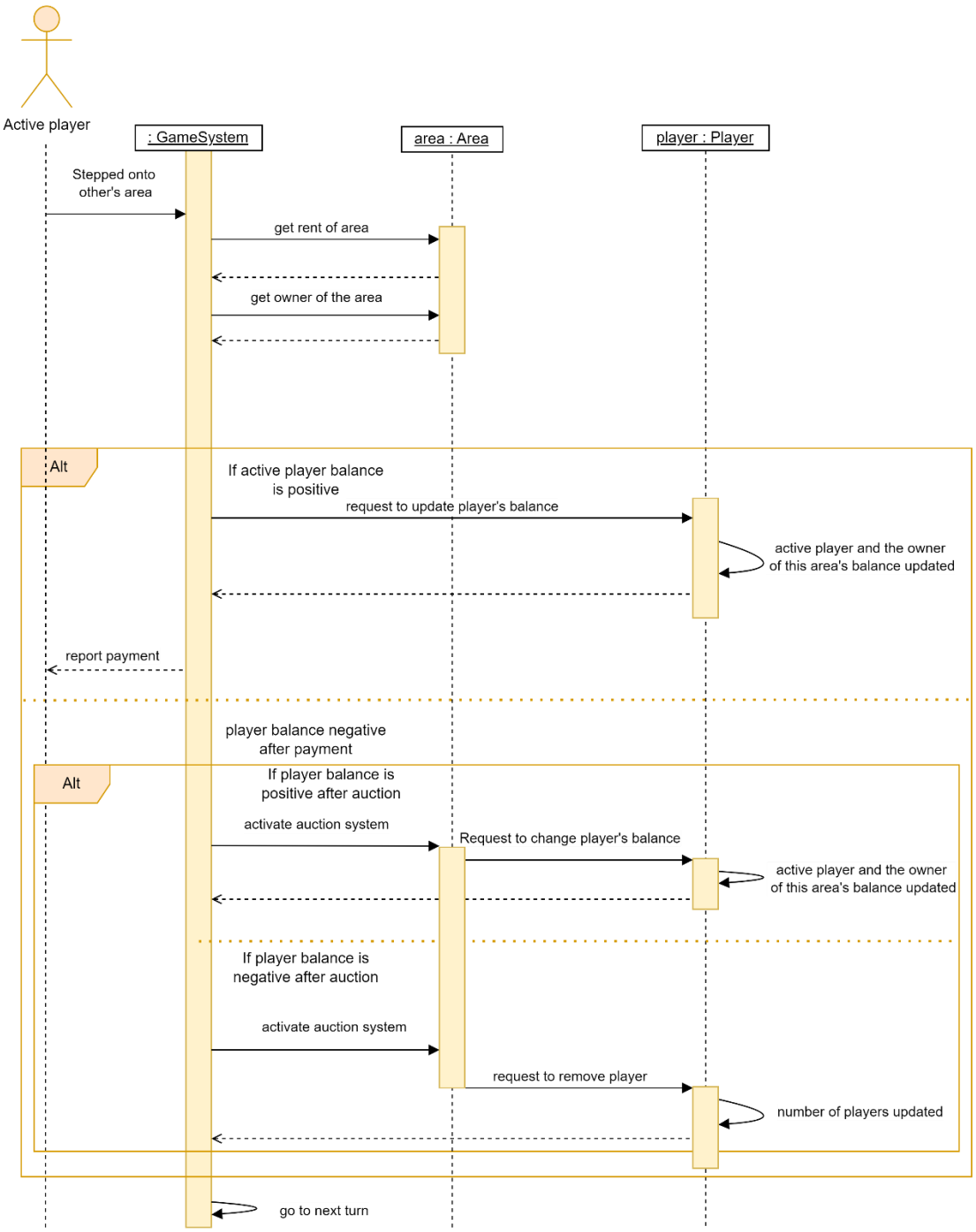
For each player registered, the system loops until a valid unique name is entered, which enables the creation of a new Player instance, whose reference is stored in the players ArrayList in GameSystem.



Roll dice and move

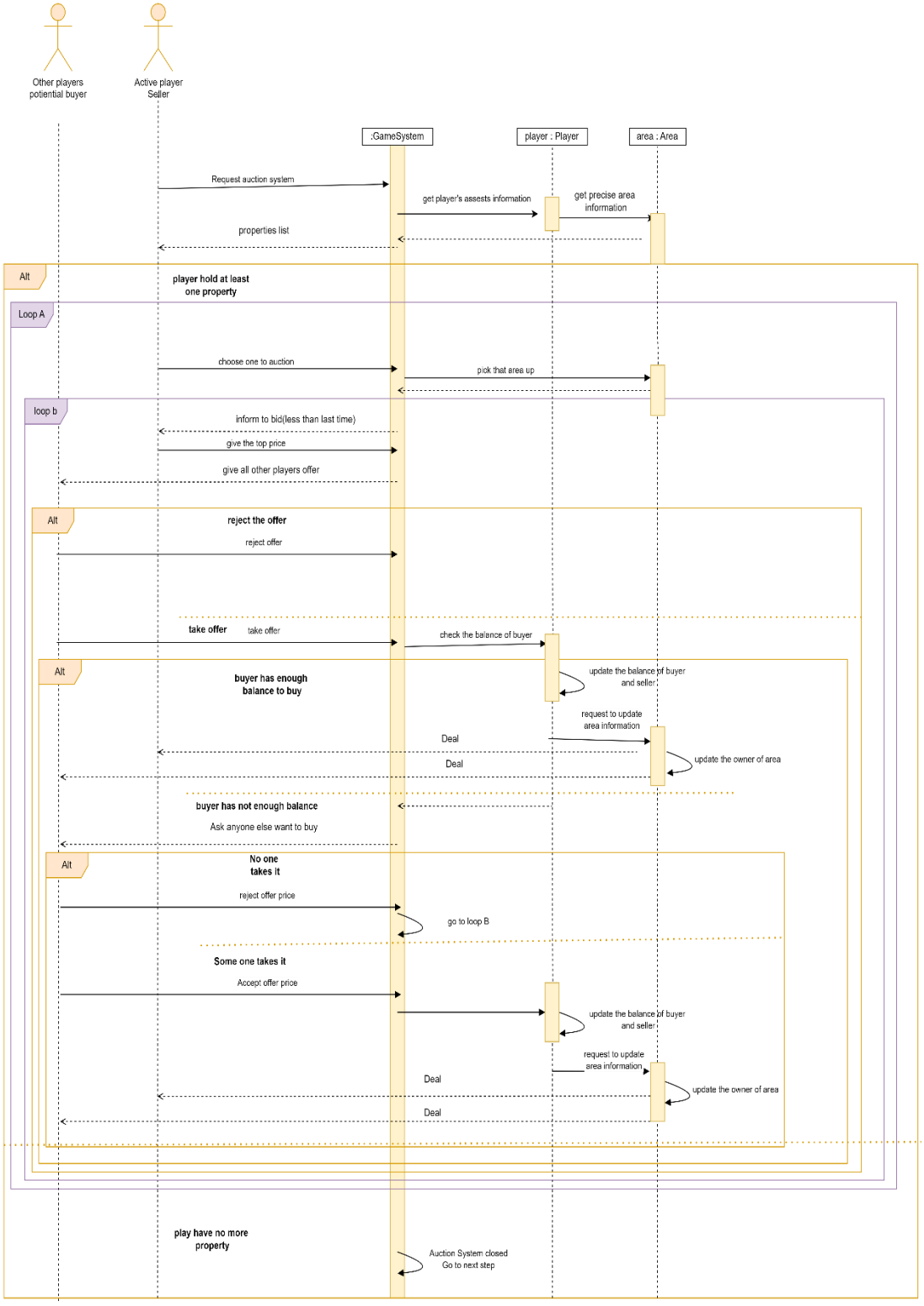
In this Sequence diagram we explain how the move method works. There's a reference block that calls a generic object that is part of the Square parent class. This object represents in the diagram a generic object as the Area, Holiday or Sunrise square. We can have many different child classes of square and we reference the specific act method in a different sequence diagram. For example, the act of Area triggers the purchase area sequence diagram that is represented by another sequence diagram. The purchase area is a method in the Area class that is a subclass of Square.

Rent payment

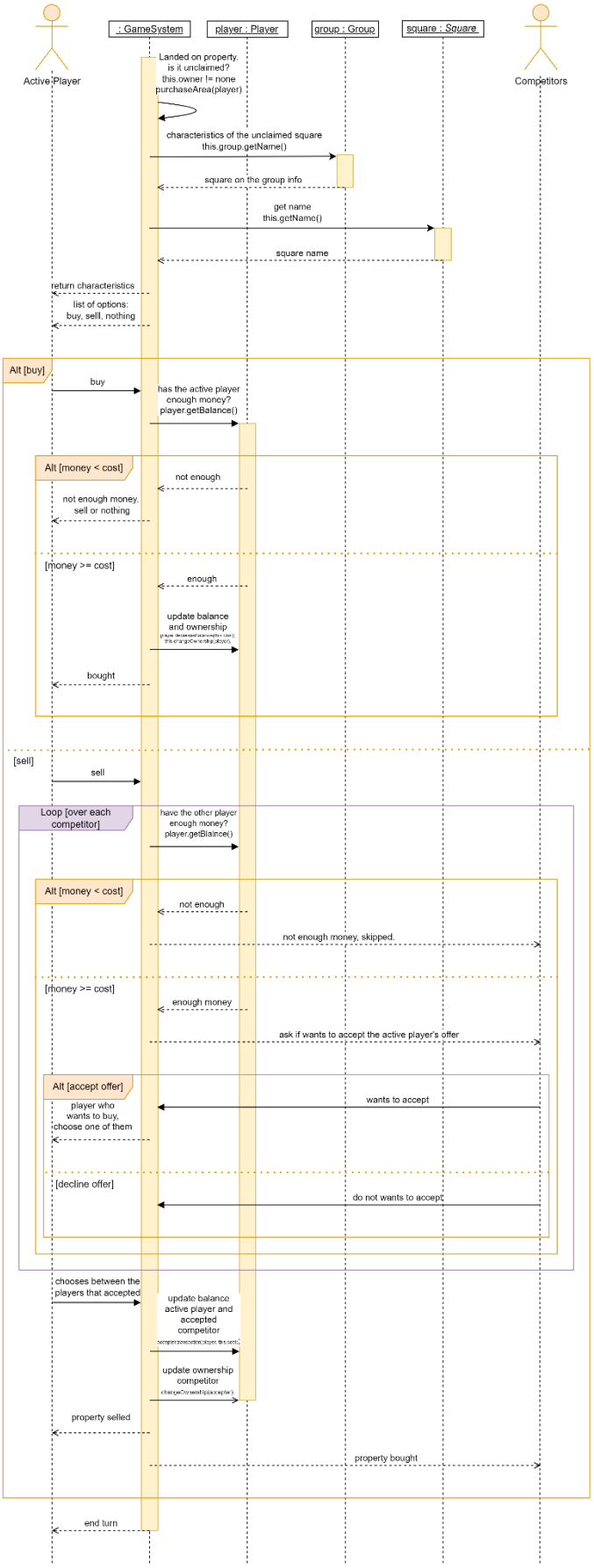


In the sequence diagram of rent payments, it shows how rent payment works automatically in game. There are in total three classes included in this diagram, the rent payment method was written in the Area class. If player does not have enough money to pay, then the Dutch auction system will be revoked. In the event where after trading this player still does not have sufficient funds, they will be removed from the game and can no longer win.

Dutch Auction System

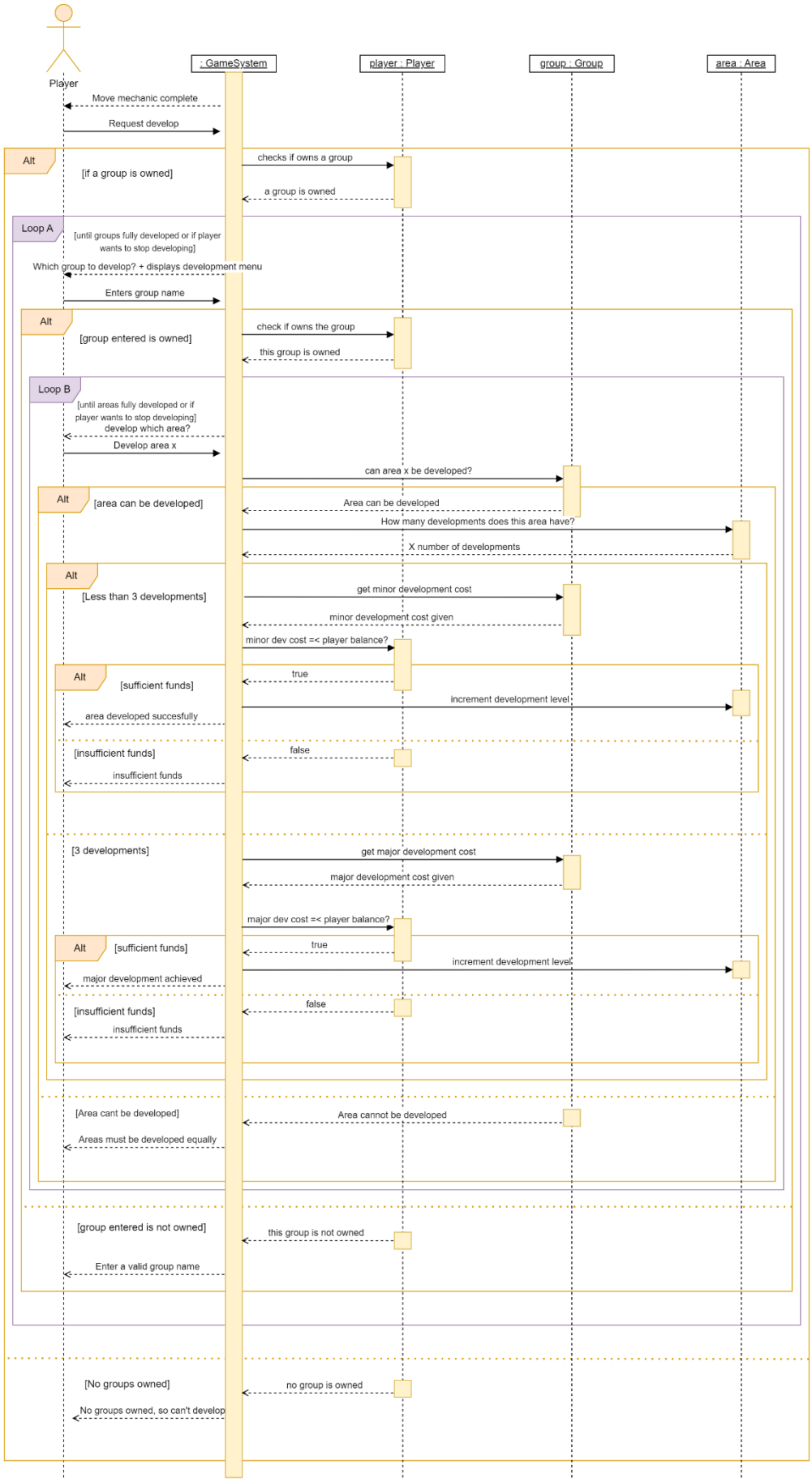


In the sequence diagram of Dutch auction system, there are two characters, one is the active player whose turn it is, and the other character is the potential buyer. The auctioneer will keep offering a descending price until someone buys it. The auction system will remain open until player chooses not to auction more properties or if there are no more properties to auction.



Purchase area

This sequence diagram describes all the interactions between the system and 2 actors, in our case one actor is the Active player and the second one is the Competitor. They are both needed to explain the process of transferring the property to another player when the square is without owner, as it was asked in the requirements. The Active player has a series of options (buy, sell, nothing). Once the sell option is triggered the second actor (competitor) should interact with the GameSystem alongside the active player to complete the required choices and allow the system to process the options.



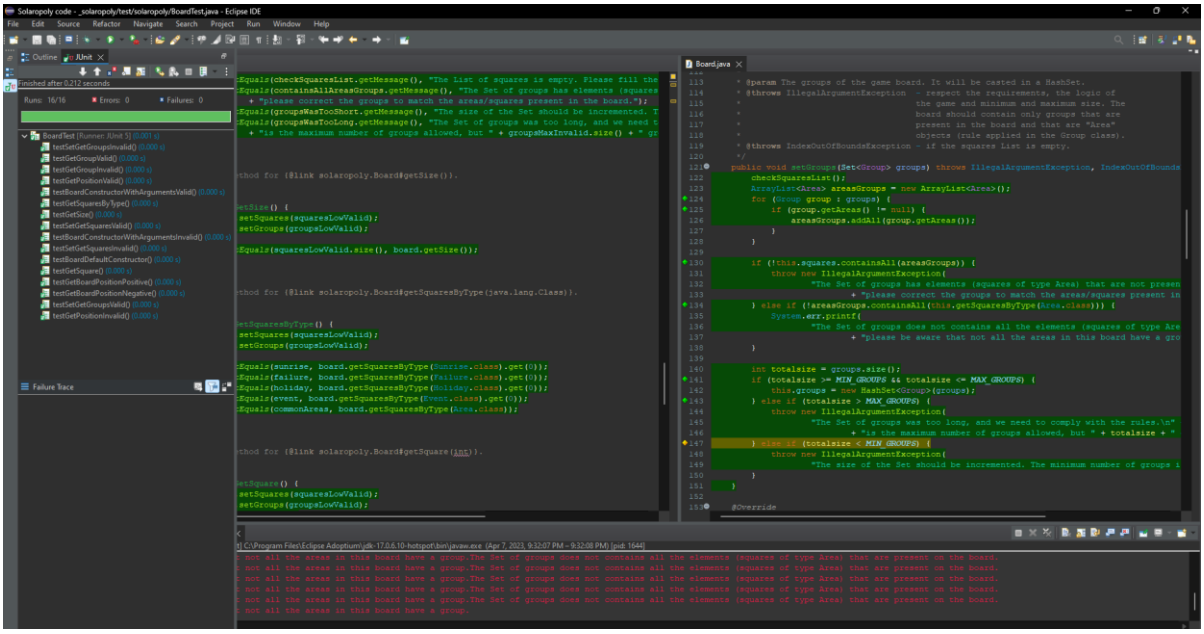
Develop area

When a player requests development at the end of their turn, the system checks if they own a whole region/group. If so, it asks which region should be developed and which area. These choices are validated based on player balance and development level.

Appendices

I. Test Plan

JUnit test board class



JUnit test of all the methods and business rules of the board class

The JUnit test conducted was able to show us if all the business rules of the board class were working correctly and all the methods were working with “boundary values testing “(minimum invalid, minimum valid, middle value, maximum valid, maximum invalid). The tested business rules allowed us to check if all the areas of the board match with the one of the groups, if there were still squares of the type of Area without a group, limits on the size of squares in the board, of the number of groups etc. We also tested negative and positive values for the getBoardPosition and checked if the number was correct even if the rollDice was giving us a number bigger than the board itself, if the numbers of time the player passed Sunrise square where correct for each loop on the board and the returned square + position. We also tested all the errors like IllegalArgumentException, IndexOutOfBoundsException and if the error message was matching too.

Acceptance test

To construct a test plan, an acceptance plan was created to identify core concepts that needed to be satisfied in the game.

Acceptance Plan for Virtual Monopoly Style Game

Objective: The objective of this acceptance plan is to ensure that the virtual monopoly style game developed meets the requirements outlined in the project scope and is fully functional and usable.

Acceptance Criteria:

Game Setup:

- The game should allow up to four players to be entered at the beginning of gameplay.
- Players should be able to choose their names.
- The game should assign a random turn order to the players.

Gameplay:

- The players should be able to throw two virtual dice at a time during their turn.
- Players' pieces should move on the board.
- Players should be able to view where they have landed and their obligations or opportunities.
- Players should be able to make choices about their actions.
- The game should indicate when a player's resources have changed and announce their new balance.
- The start or "Sunrise" square should enable players to pick up their resources.
- There should be a square on the board where nothing happens.
- The game should have four regions.
- Two of the regions should consist of three areas each.
- Two of the regions should consist of two areas each.
- Before a player can develop an area within an area, they must own the whole region.
- Players should be able to develop areas in regions that they already own.
- Three developments are needed before a player can establish a major development.
- When a player lands on an area they do not own, they must give up some resources.
- When a player runs out of resources, the game should display the amount of resources each player holds.

End of Game:

- The game should end when a player no longer wants to play.
- The game should end after 50 turns.
- The game should end when the target output is reached before 50 turns.

Testing:

- The game will be tested to ensure that it meets all of the acceptance criteria outlined above.

Usability:

- The game will be tested for usability to ensure that it is easy to use and understand.

Deliverables:

- A fully functional virtual monopoly style game that meets all of the acceptance criteria outlined in this plan.

Test Plan for Virtual Monopoly Style Game:

A number of test cases needed to be fulfilled for the test plan to meet the acceptance plan. These included:

1. Testing player functionality: Ensure that up to four players can be added to the game, and that their names are correctly entered and displayed during gameplay.

2. Testing dice functionality: Ensure that the virtual dice generate random numbers between 1 and 6, and that players correctly move the number of spaces indicated on the dice roll.

3. Testing player obligations and opportunities: Ensure that players are correctly informed of their obligations and opportunities when they land on specific squares, and that the correct actions can be taken.

4. Testing resource functionality: Ensure that changes to a player's resources are correctly tracked and displayed throughout the game.

5. Testing square functionality: Ensure that players correctly collect resources when they pass the "Sunrise" square, and that nothing happens when they land on a specific square.

6. Testing field functionality: Ensure that fields are correctly defined and that the required conditions for developing areas within a field are correctly implemented.

7. Testing development functionality: Ensure that development can be correctly established within a field, that the correct cost is applied, and that players can correctly pay or invest in developments.

8. Testing resource consumption: Ensure that the correct amount of resources is subtracted when players land on and develop areas within a field.

9. Testing end game functionality: Ensure that the correct outcome is reached when a player runs out of resources or chooses to end the game.

10. Testing overall game functionality: Ensure that the game can be played from start to finish without any major bugs or errors, and that the overall gameplay experience is enjoyable and intuitive.

II. Team Minutes

Date: 7 Feb 2023
Location: One Elmwood
Objectives: Meet for the first time, review the specification and discuss how to approach the task
Attendees: Andrew, Dan, Li, Roberto, Sam
Apologies: -
Agenda: None
Discussions (challenges, and progress by each team member on previous tasks):

- We are discussing about the timing for each cycle, it should be 1 week
- We discussed about the testing part, we are thinking that it's more practical to test the whole class instead of testing the single method and then test the whole system, we want to follow the V model by starting from a single class
- We are thinking about using generalization for the actor in our UML model, in our case the player (4 of them) could be many actors at the same time (one could be the owner of the building and another one the guy who pays the rent), but it's too messy and complex in this case so we could generalise it
- We decided to build a waterfall model for the steps we need to follow for our project
- How to build each case, how specific it should be
- We decided to decide and ask advice for how to split the work in the group, we talked about some ideas
- We talked about the requirements, how to build them, how to prioritise them and what path should be followed to build them (with use cases and stories)
- Talking about use cases and deciding if we need to build use stories to help in the creation of these, we decided that could be helpful in particular cases but not necessary for other situations
- Going through the requirements, decide which resource use instead of cash
- Talking about the cloud environment we'll use to work on the project
- Talking about the user interface if it's better to use it or not for us to understand if the game respect the requirements (validation)
- Choice of the model to use with some variations (scrum)
- Discussion about the requirements and the initial design of the project
- Talking about the framework and the programming language to use

Action Plan (with allocations to each team member):

- **All** to absorb spec, continue discussions on WhatsApp and agree next meeting date

---

Date: 20 Feb 2023
Location: One Elmwood
Objectives: Formalise our requirements analysis process
Attendees: Dan, Li, Roberto, Sam
Apologies: Andrew couldn't attend due to illness
Agenda:

- Set up project report document to work in
- Check use case diagram and start sketching use case descriptions
- Agree week's work – writing out use case descriptions and maybe system design (Sequence + Class diagrams)
- AOB

Discussions (challenges, and progress by each team member on previous tasks):

- We continued to work out the mechanics of the game. Li led work on a flowchart. We attempted to start work on Sequence Diagrams but decided these and Class Diagrams

would be best developed alongside prototyping in Eclipse. We switched back to a flowchart / Activity Diagram format to sketch out the mechanics, which was more successful.
- Discussion seemed to basically validate the 4-ellipse Use Case Diagram developed by Sam over the previous week, but with at least one additional use case, "Cancel Registration", and we will keep it under review especially as we add functionality beyond the specification.
- We agreed to meet again as soon as Andrew is well to play Monopoly and discuss further.
- Activity Diagram for moving on the board shared with Andrew via WhatsApp.

Action Plan (with allocations to each team member):
- **All** continue to absorb spec as needed.
- Use case description work to be planned next time – **Sam** commits to do an example.

Date: 21 Feb 2023
Location: Sam's place
Objectives: Play Monopoly, eat pizza, progress requirements analysis, check out Kanban and GitLab
Attendees: All
Apologies: -
Agenda:
- Play Monopoly to get an intuitive sense of this type of game and see all the required mechanics for the project in action (as well as others we may wish to add)
- User stories and scenarios – where do they fit in?
- Continue work on requirements analysis, planning to complete use case descriptions soon, allocating a member for each use case description.
- The "Save Our Planet" metaphor – any creative ideas?
- AOB

Discussions (challenges, and progress by each team member on previous tasks):
- We played Monopoly to get a better understanding of the mechanics of the game.
- Talked about the idea of splitting the 5 possible user interactions in Sam's use case diagram between the 5 members of the group while being open to the idea that more may arise as we progress.
- Discussed the possible assets that may be used as currency in the game. Suggestions included Carbon credits or H_2
- We have created a hypothetical story for our game. The game is set in a dystopian world where pollution levels have reached their limit. In this society, businesses are required by law to reduce waste and pollution, prioritize efficiency, and use clean energy to achieve the zero-pollution goal. The government awards carbon credits to those who comply with these objectives and imposes taxes on those who do not. To win the game, it is essential to have enough credits to avoid the risk of inspectors closing down the business. We have also discussed the possibility of having two assets: profits and tradable carbon credits. The winner of the game will be the player who doesn't go in negative profits or the player who does not run out of carbon credits.
- Decided that trade between users was not part of the project specifications but we may still add it further down the line.

Action Plan (with allocations to each team member):
- Roberto Lo Duca will work in the use case description and develop the sequence diagram

---

Date: 27 Feb 2023
Location: Room in CompSci building
Objectives: Agree on uses cases and finalising use case diagram, allocate use case descriptions
Attendees: Andrew, Sam, Li, Roberto
Apologies: Dan

Agenda: None
Discussions (challenges, and progress by each team member on previous tasks):
- Roberto Lo Duca had run 4 candidate use case diagrams past Zheng.
- Debated and firmed up use cases and use case diagram based on Zheng's advice

Action Plan (with allocations to each team member):
- Roberto Lo Duca and Li to complete their use case descriptions
- Ask Zheng tomorrow about exception flow and view info case (how should it be connected)

Chat with Zheng, 28 Feb
- Use case diagram is ok? *Yes, just change A/B to active/competitor and add Reject unowned property as a use case.*
- Should we build a different use case for displaying different information like: you passed GO, Balance, position, property etc?
- Requirements discovery has been done for us but if we add features should we develop those formally with user stories, scenarios, requirements? *Zheng recommends user stories and requirements but didn't seem to be recommending an exhaustive process with scenarios.*
- How should we structure our development of features beyond the core requirements – we are thinking of a 'core' use case diagram that we finalise now and an 'expansion' use case diagram that we build on with later cycles of discovery, analysis, design and implementation. *Zheng said the core diagram would be valid as a subset of the entire final diagram. He seemed fine with the idea of discovery, analysis, design carrying on and adding to the sprint backlog, as long as we prioritise appropriately. Each sprint should end with a playable game even if it's lacking big mechanics.*
- Should the design phase be intuitive or is there a process to follow (beyond class and sequence diagrams)? *Seems like we should just talk a lot and document our design decisions in the report.*
- Should our use cases map 1:1 with sequence diagrams or might we combine related use cases in one sequence diagram? *In general, a sequence diagram will cover one use case but seems ok to combine if it makes sense. Also – Zheng foresees sequence diagrams being a lot of work and recommends we only do diagrams for the essential requirements, not for any others we discover.*

Date: 28 Feb 2023
Location: One Elmwood
Objectives: Review Zheng's advice and get into the design phase
Attendees: All
Apologies: -
Agenda: As objectives
Discussions (challenges, and progress by each team member on previous tasks):
Action Plan (with allocations to each team member):
Sequence diagrams needed (priority):
- Register player – Sam
- Take turn/roll dice/move – Dan
- Buy unowned area – Roberto Lo Duca
- Develop area – Andrew
- Pay rent – Li

Date: 4 Mar 2023
Location: Teams
Objectives: Review our progress on sequence diagram and discuss class structure

Attendees: All
Apologies: -
Agenda: As objectives
Discussions (challenges, and progress by each team member on previous tasks):

- We reviewed each member's sequence diagram, developed our understanding of how the classes would work in practice, and agreed to carry on working on them for Monday.
- Sam suggested that the game would run on a single main thread in the GameSystem class, setting up instances of squares, groups (fields), and players (pieces) – and that development, purchasing etc. would be functions of and on those classes/objects.

Action Plan (with allocations to each team member):

- All to converge structure diagrams towards a single class structure.

Date: 7 Mar 2023
Location: CSB Lab
Objectives: Set up bare bones game system and discuss GitLab and Java
Attendees: all
Apologies: -
Agenda:

- GameSystem
- GitLab
- AOB

Discussions (challenges, and progress by each team member on previous tasks):

- All have drafted sequence diagrams
- Sam updated class diagram based on meeting with Zheng

Action Plan (with allocations to each team member):

Date: 16 Mar 2023
Location: Teams
Objectives: Start sprint
Attendees: Sam, Roberto Lo Duca
Apologies: Andrew, Li, Dan
Agenda: As objectives
Discussions (challenges, and progress by each team member on previous tasks):
How do we coordinate??
Action Plan (with allocations to each team member):

- Roberto Lo Duca: Board, Player and purchaseArea()
- Sam: Group, Square family and GeneratesIncome, registerPlayers()
- Coordinate with others about other aspects – suggest they do code related to their sequence diagrams

---

Date: 20 Mar 2023
Location: Teams
Objectives: Mid-sprint review – discuss latest advice from Zheng
Attendees: Andrew, Li, Roberto Lo Duca, Sam
Apologies: Dan
Agenda:

- Make sure everyone has at least one task they can contribute to in this sprint
- When shall we finish the sprint? (Weekend at the latest)

Discussions (challenges, and progress by each team member on previous tasks):

- Roberto Lo Duca had got further advice from Zheng about sprint tasks. We need to set deadlines for each sprint. Divide the job for each member in classes or methods it's a good strategy. The team reviews the user stories and breaks them down into specific tasks that needs to be completed. The tasks are then prioritized and assigned to each team member. Each method should be refactored or encapsulated inside the correct class. So, Zheng advised to think where to move the method "develop area" and "purchase area" (he advised to move it in the square class for example). We have plans to discuss together the class diagram again with Zheng. Zheng also told us that we can modify the class diagram later during the sprint or after. Another advice was to work on the essential classes at the first sprint and after it consider adding optional features like "cards", "trade" etc.
- Sam had set up Planner board and completed Group, Square family, GeneratesIncome, setNumPlayers(), registerPlayers()
- We developed the task board and noted a required order of development:
    - code setting up the board
    - code setting up the turn, roll and move on the board
    - the act() method of Area needs to allow purchase or passing on of an area (if it's unowned) or paying rent (if owned)
    - we need to be able to purchase areas before we can pay rent or develop areas

Action Plan (with allocations to each team member):

- Andrew, Li and Dan to set up GitLab

Date: 30 Mar 2023
Location: Parlour
Objectives: Review sprint, plan deliverables
Attendees: All
Apologies:
Agenda:
Discussions (challenges, and progress by each team member on previous tasks):
Action Plan (with allocations to each team member):

- **Andrew, Roberto Lo Duca, Li** to coordinate on bug fixes
- **Dan** to work on Acceptance Plan (Appendix I)
- **Sam** to compile Appendices II-IV
- **Sam** to add sequence diagram for setting number of players
- **Li** to finalise sequence diagram for paying rent

III. Git Lab evidence

Contributor commit graphs

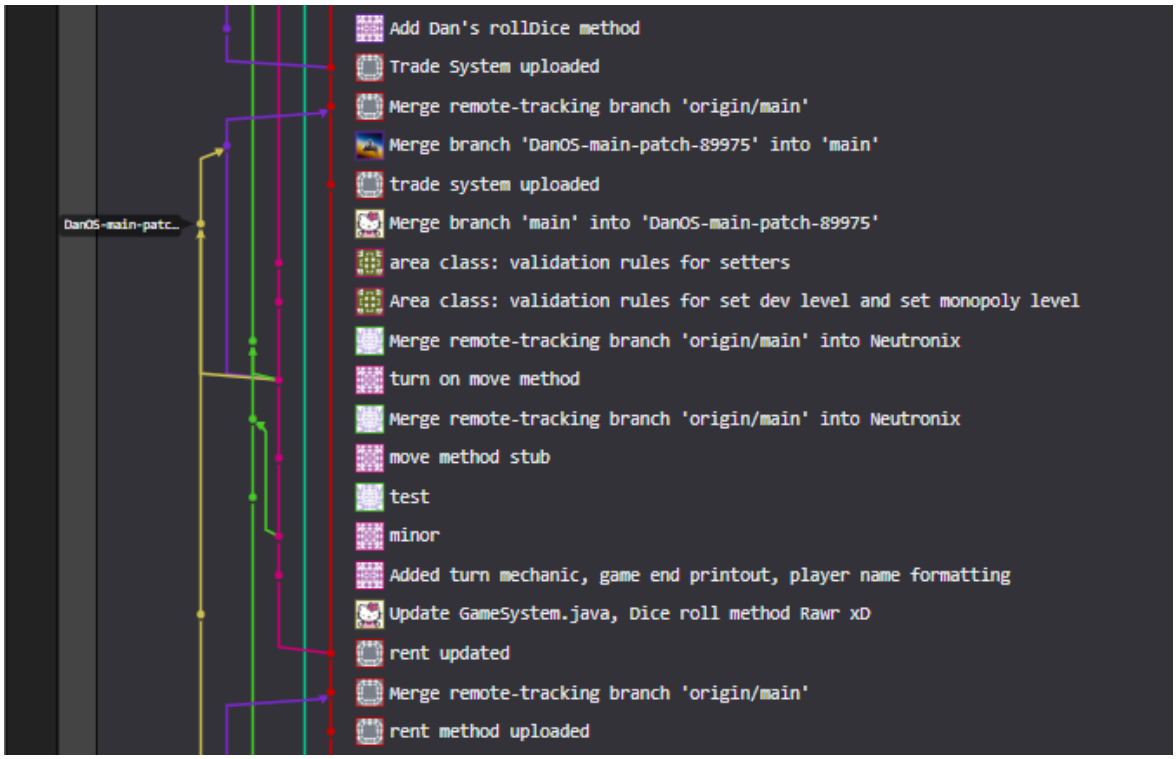
Each team member contributed to the Git repository from one or two email accounts:

- Andrew: andrewscott
- Dan: dosullivan08
- Li: litterpigger, hli50
- Roberto: robertoloduca
- Sam: smillar36, samillar94

The contributor commit graphs for the main branch (into which all branches were merged by the end) are shown below.

Main branch commit graphs by contributor

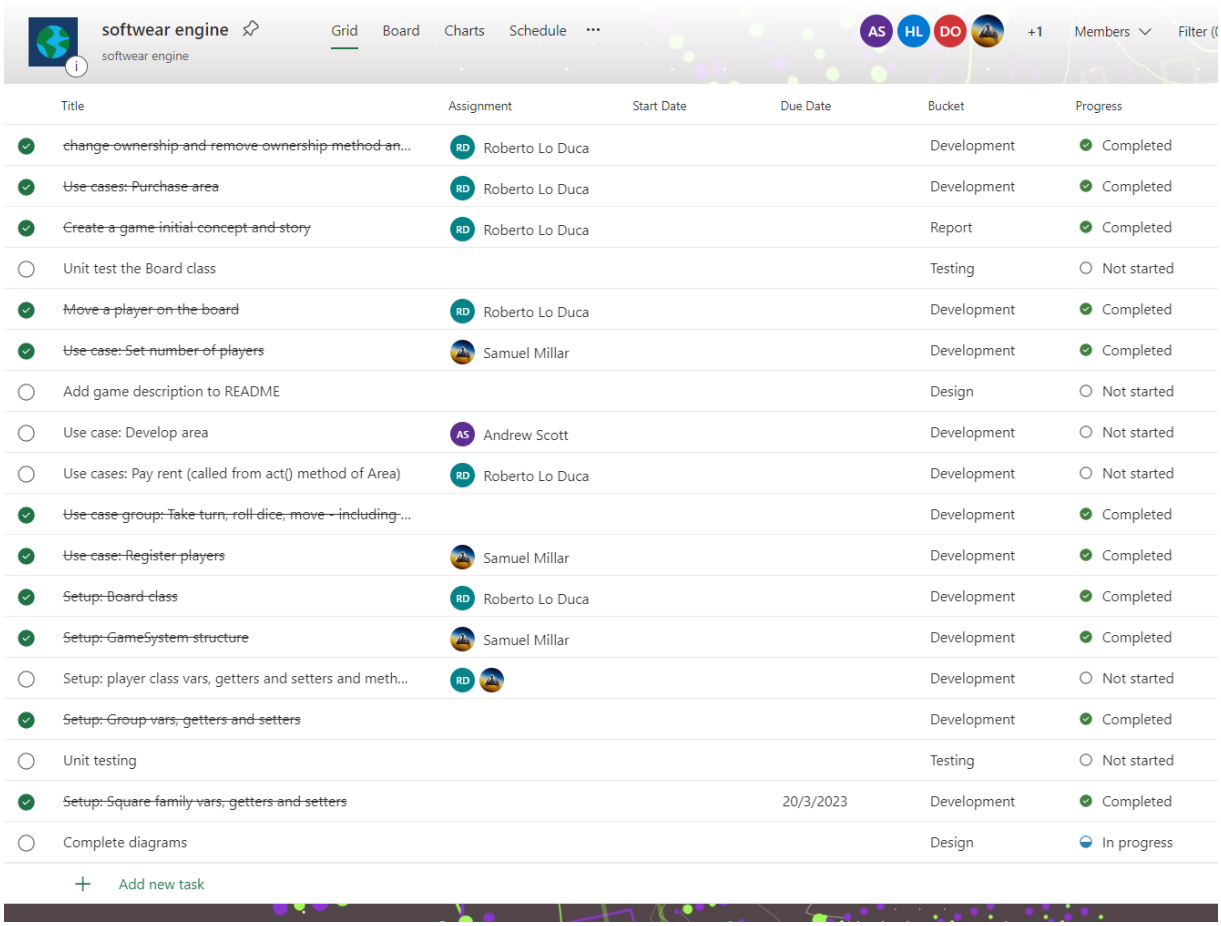
Sample branch graph



IV. Project management evidence

We followed a Scrum model in our system implementation, setting up a backlog in Microsoft Planner which each team member could pick tasks from. Our first sprint aimed to produce a playable game with all of the required functionality present, and later sprints finished off core tasks which proved difficult initially, as well as enhancing input validation and user experience.

Sample sprint backlog from Sprint 1



softwear engine

softwear engine

Grid Board Charts Schedule ···

AS HL DO +1 Members Filter

Title	Assignment	Start Date	Due Date	Bucket	Progress
~~change ownership and remove ownership method an...~~	RD Roberto Lo Duca			Development	Completed
~~Use cases: Purchase area~~	RD Roberto Lo Duca			Development	Completed
~~Create a game initial concept and story~~	RD Roberto Lo Duca			Report	Completed
Unit test the Board class				Testing	Not started
~~Move a player on the board~~	RD Roberto Lo Duca			Development	Completed
~~Use case: Set number of players~~	Samuel Millar			Development	Completed
Add game description to README				Design	Not started
Use case: Develop area	AS Andrew Scott			Development	Not started
Use cases: Pay rent (called from act() method of Area)	RD Roberto Lo Duca			Development	Not started
~~Use case group: Take turn, roll dice, move - including ...~~				Development	Completed
~~Use case: Register players~~	Samuel Millar			Development	Completed
~~Setup: Board class~~	RD Roberto Lo Duca			Development	Completed
~~Setup: GameSystem structure~~	Samuel Millar			Development	Completed
Setup: player class vars, getters and setters and meth...	RD			Development	Not started
~~Setup: Group vars, getters and setters~~				Development	Completed
Unit testing				Testing	Not started
~~Setup: Square family vars, getters and setters~~			20/3/2023	Development	Completed
Complete diagrams				Design	In progress

+ Add new task