Design Document

Unit 1 Project

Bowling Alley Simulation Refactoring Project

TEAM 26

Date of Submission:-19th February,2022

TEAM INFORMATION:-

1. Husen Kagdi(2021201077)
2. Subhra Chakravorty(2021201078)
3. Akilesh Panicker(2021201081)
4. Siddharth Gupta(2021201082)

OVERVIEW:-

The original code is a simulation of a bowling game where the user is the person who is going to bowl.There is an admin of the bowling game called ControlDesk who can monitor and make changes to the functionalities of any other major classes.Here each ball throw of the user is simulated and the score for each party and lane is calculated and sent back to the user after he decides to stop playing.

The original code was fully functioning but it had some problems like redundant code,code repetitions and other code smells.These problems,if not solved may become a problem for any future developers who want to add any new functionalities or make some changes in the code design.Hence we have refactored the code and analysed the metrics of the original code and refactored code.Our refactored code attempts to reduce the load of major classes or functions by dividing them into multiple classes or modules.For example:-The functions performed by Pinsetter class was divided into two further new classes called SimulateThrow and ScoreCalculate which simulated a ball throw and calculated the score of a throw respectively.

UML Class Diagrams:-

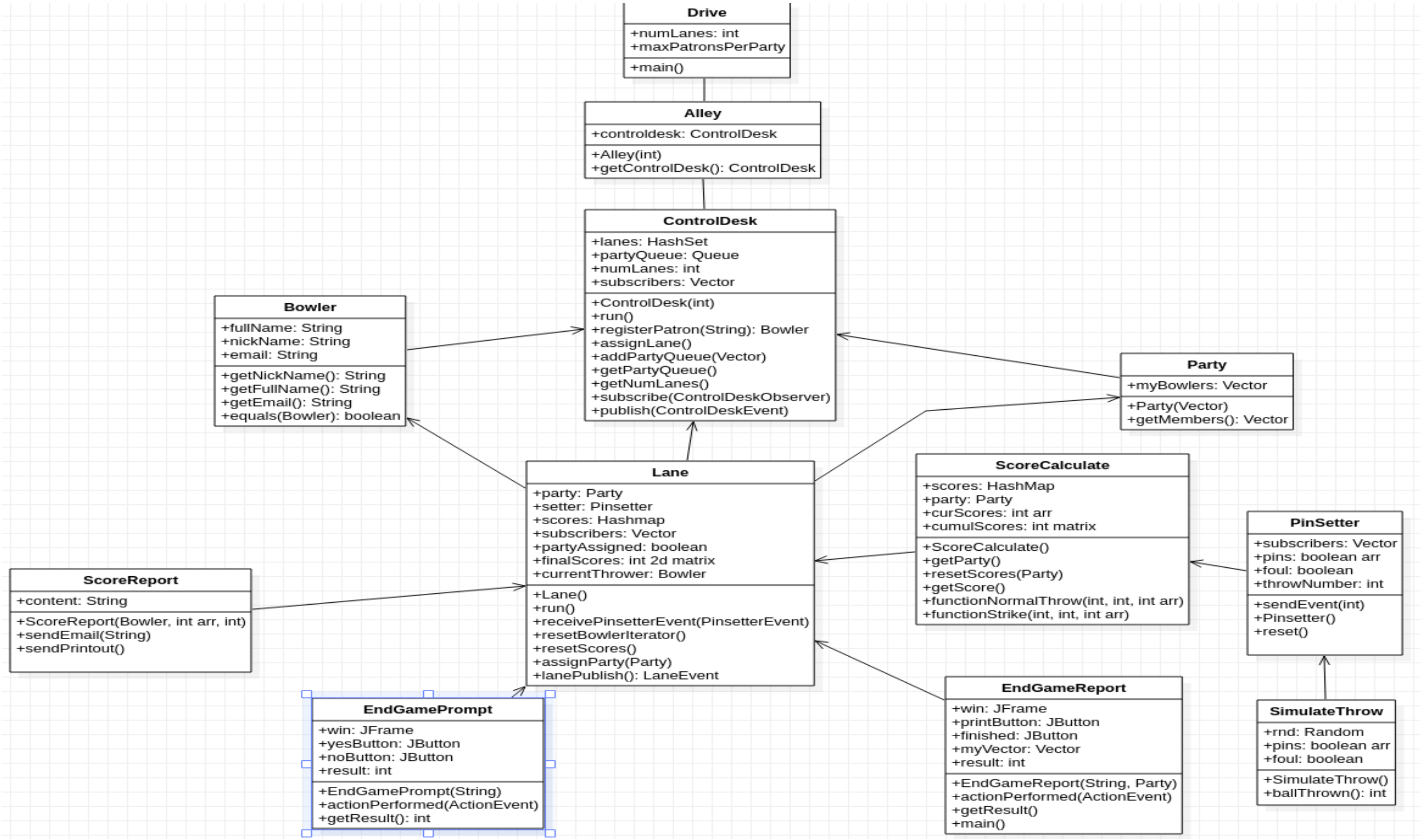

Diagram 1:-UML class diagram of major classes

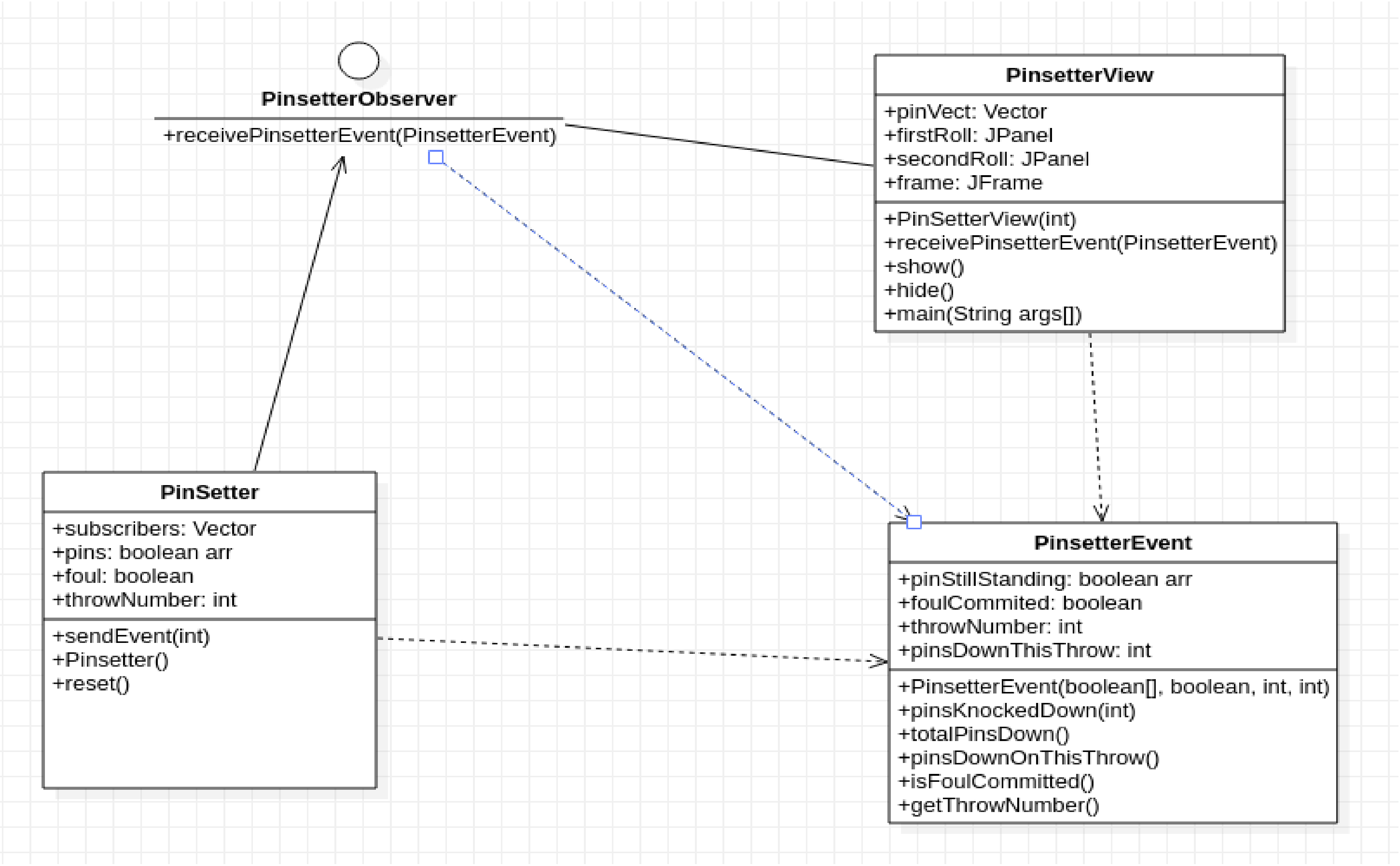

Diagram 2:PinSetter Class and its design

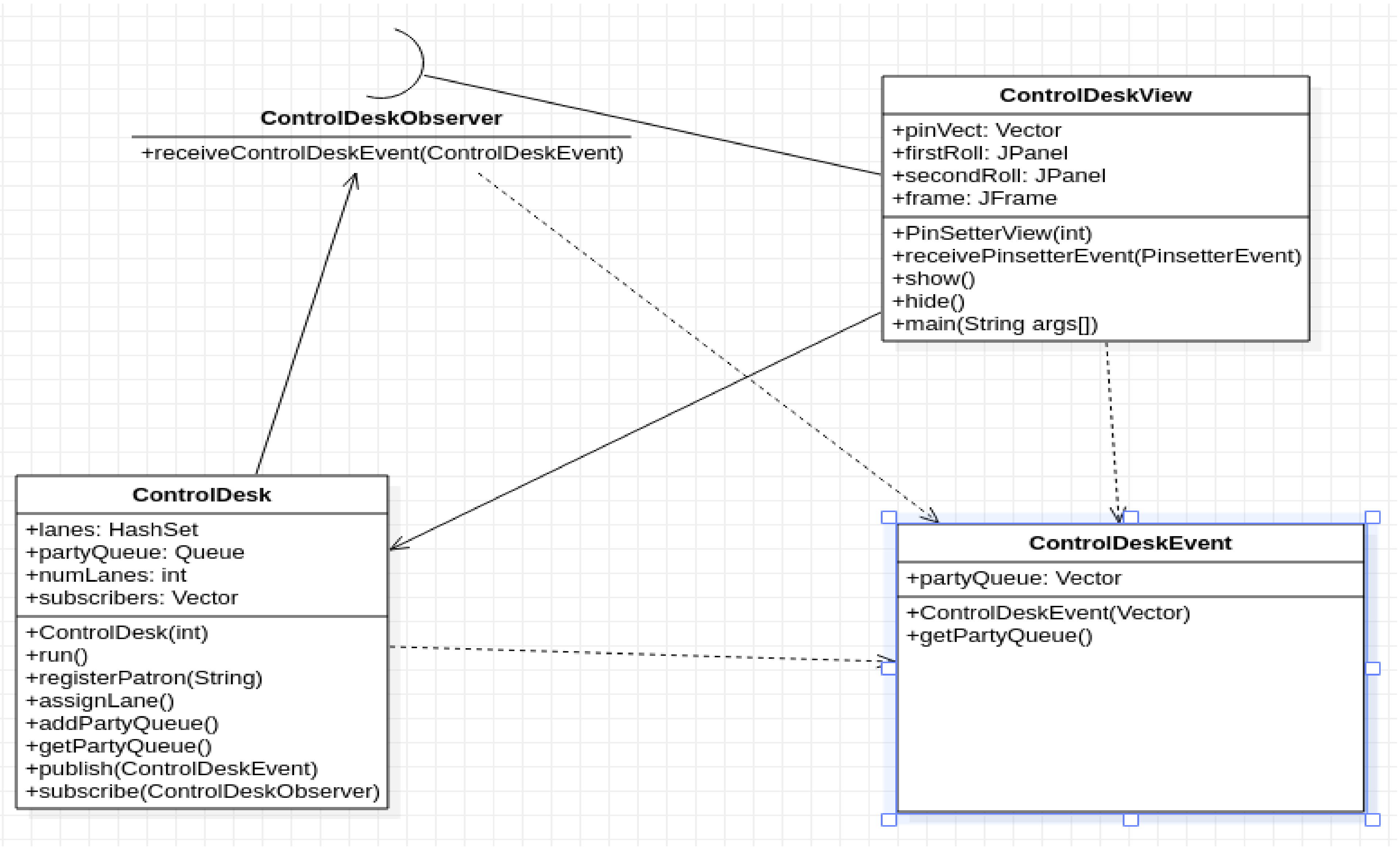

Diagram 3: Control Desk and its design

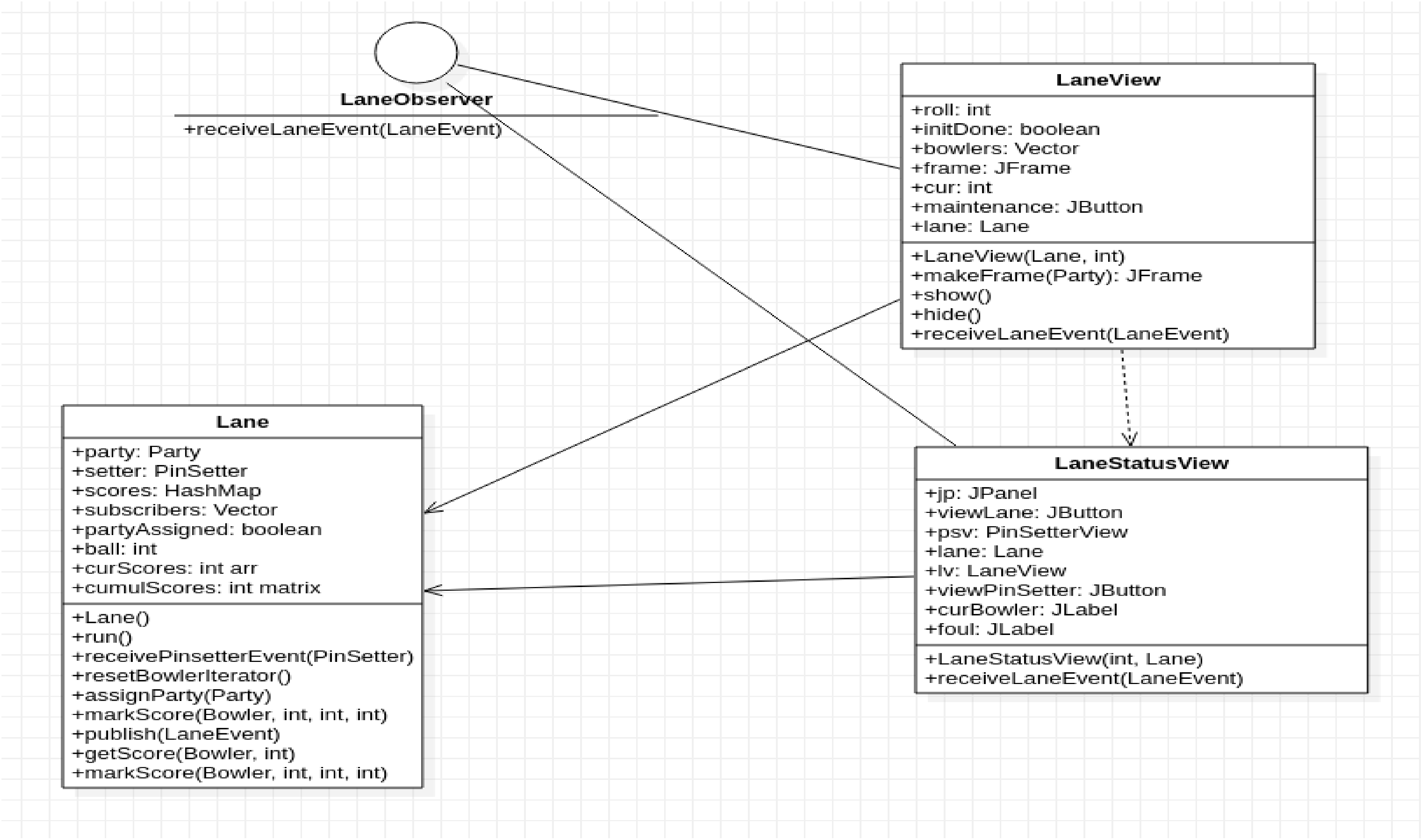

Diagram 4:Lane class and its design

Sequence Diagrams before Refactoring:-

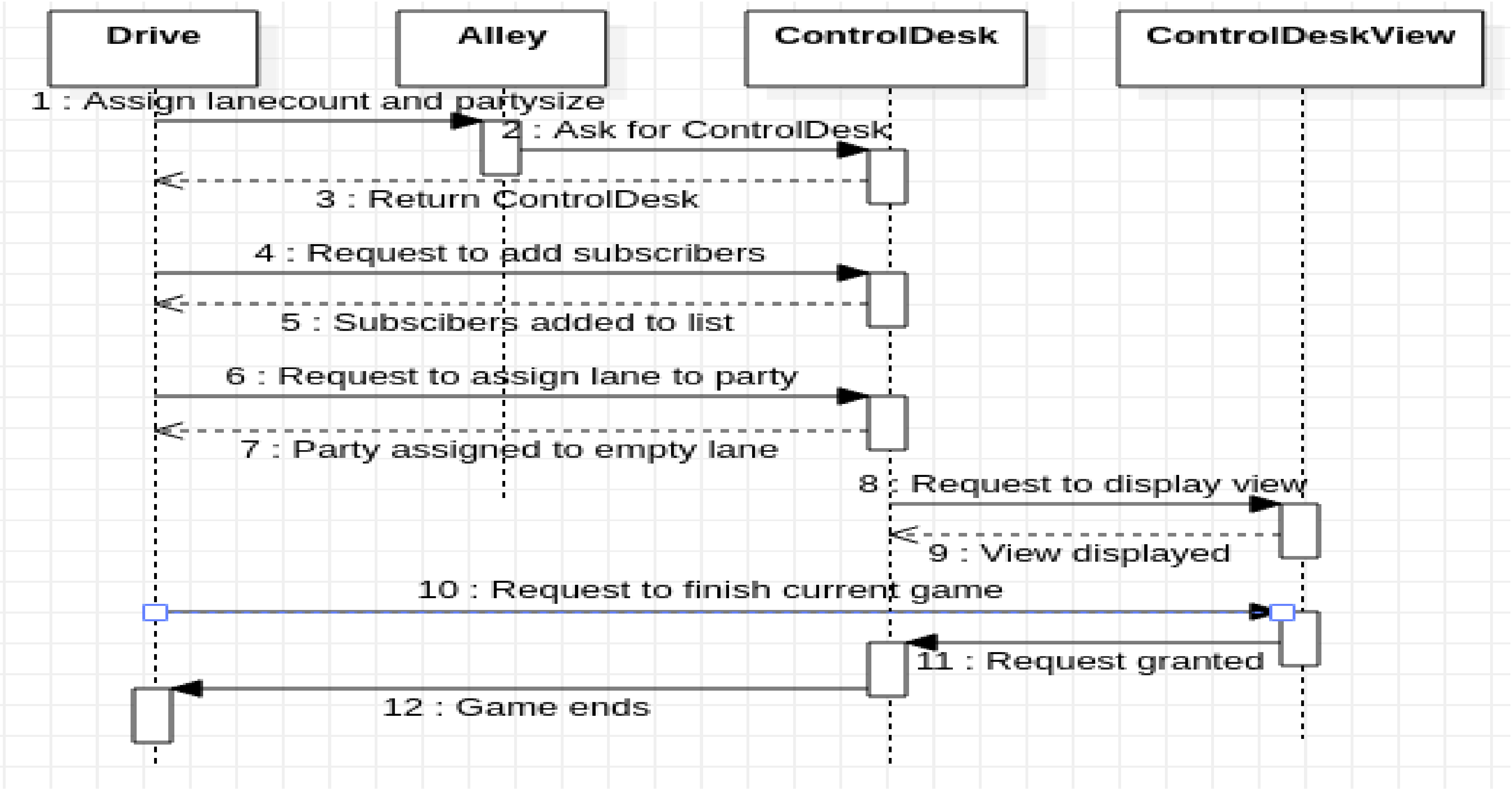
Drive
Alley
ControlDesk
ControlDeskView
1 : Assign lanecount and partysize
2 : Ask for ControlDesk
3 : Return ControlDesk
4 : Request to add subscribers
5 : Subscibers added to list
6 : Request to assign lane to party
7 : Party assigned to empty lane
8 : Request to display view
9 : View displayed
10 : Request to finish current game
11 : Request granted
12 : Game ends

Diagram 5: Assigning lane count,party size and empty lanes to party

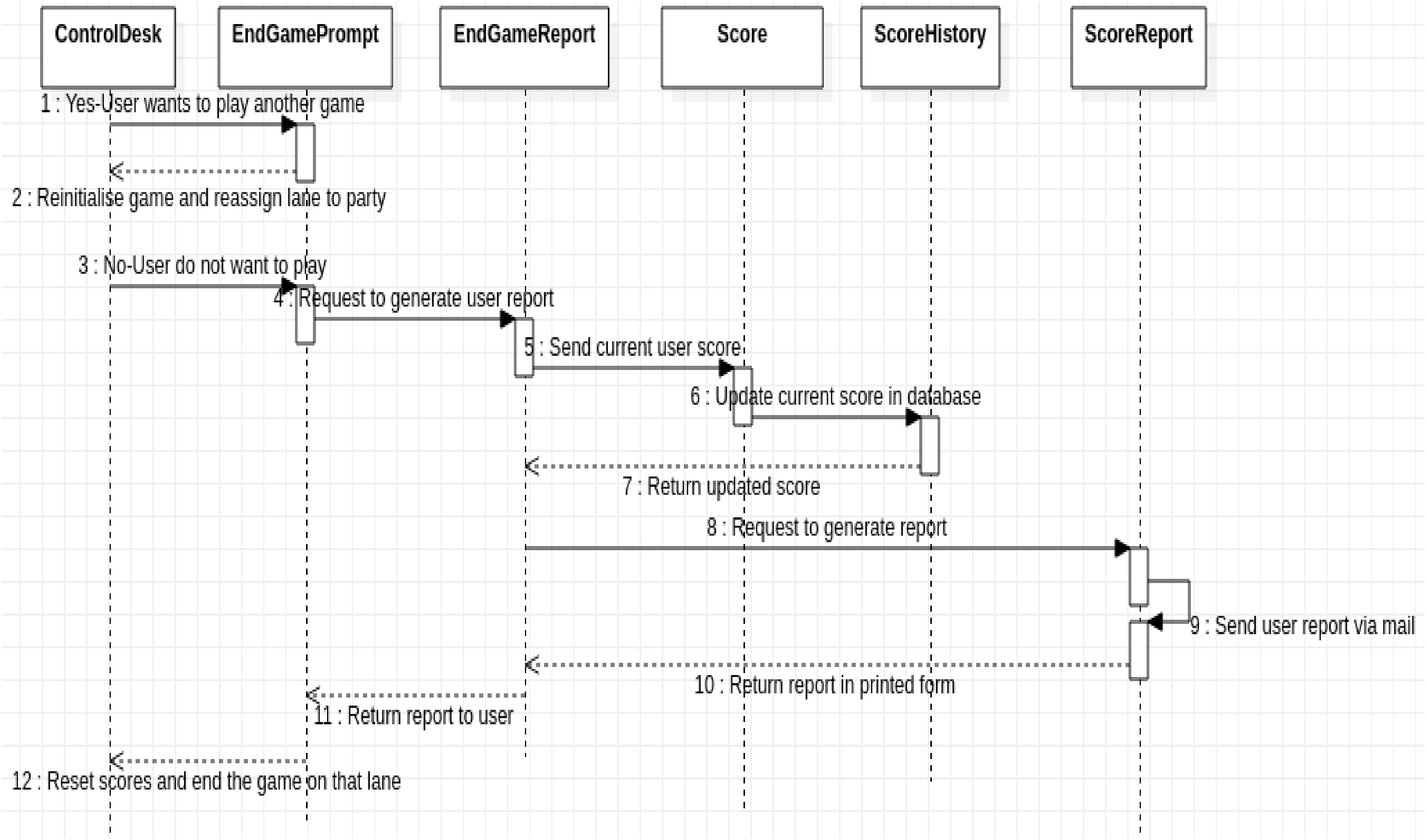
ControlDesk
EndGamePrompt
EndGameReport
Score
ScoreHistory
ScoreReport
1 : Yes-User wants to play another game
2 : Reinitialise game and reassign lane to party
3 : No-User do not want to play
4 : Request to generate user report
5 : Send current user score
6 : Update current score in database
7 : Return updated score
8 : Request to generate report
9 : Send user report via mail
10 : Return report in printed form
11 : Return report to user
12 : Reset scores and end the game on that lane

Diagram 6: Endgame prompt,score updation and report generation

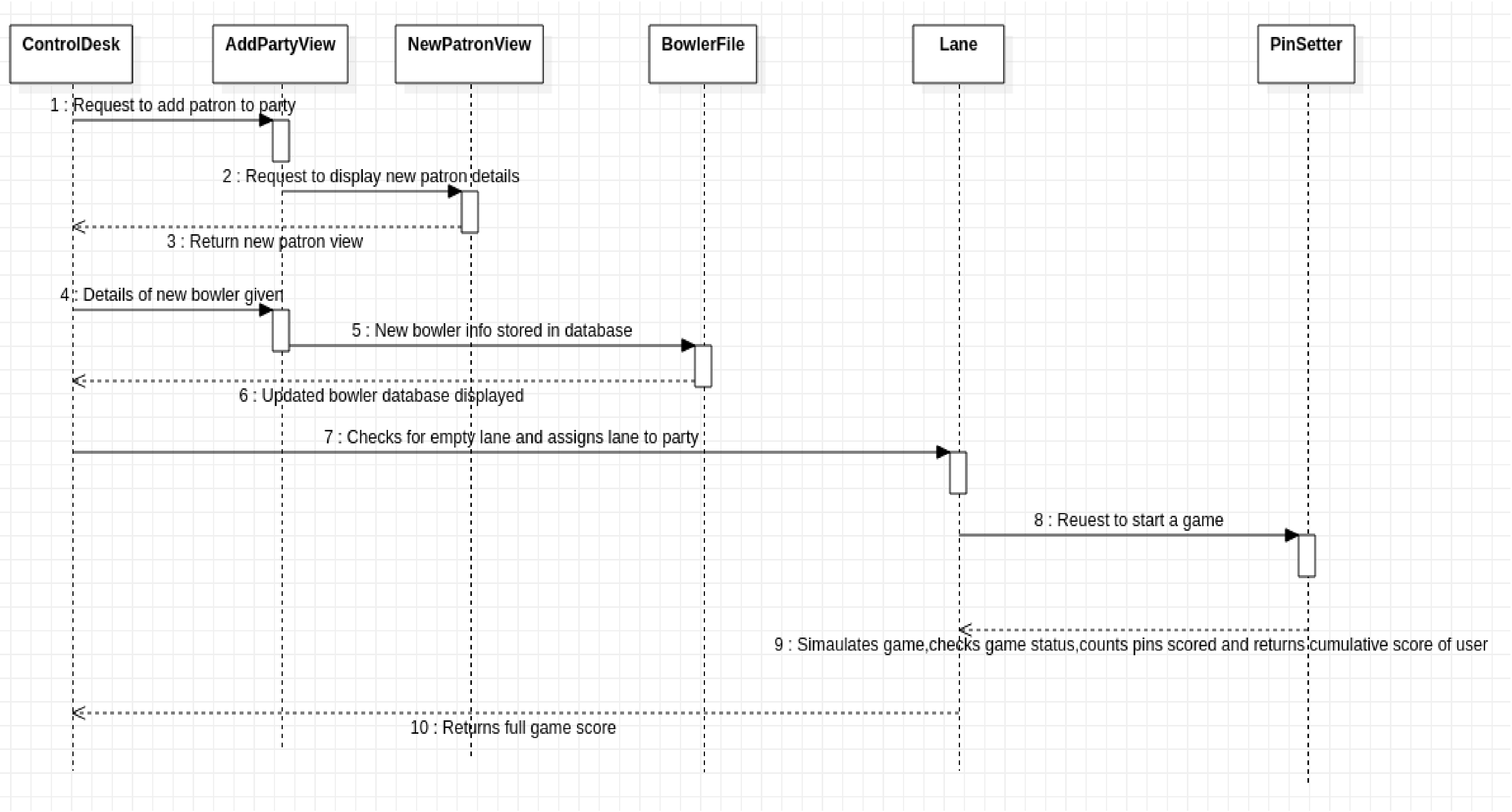
ControlDesk
AddPartyView
NewPatronView
BowlerFile
Lane
PinSetter
1 : Request to add patron to party
2 : Request to display new patron details
3 : Return new patron view
4 : Details of new bowler given
5 : New bowler info stored in database
6 : Updated bowler database displayed
7 : Checks for empty lane and assigns lane to party
8 : Reuest to start a game
9 : Simaulates game,checks game status,counts pins scored and returns cumulative score of user
10 : Returns full game score

DIagram 7: Add new user,start game and calculate score

Sequence Diagrams after Refactoring:-

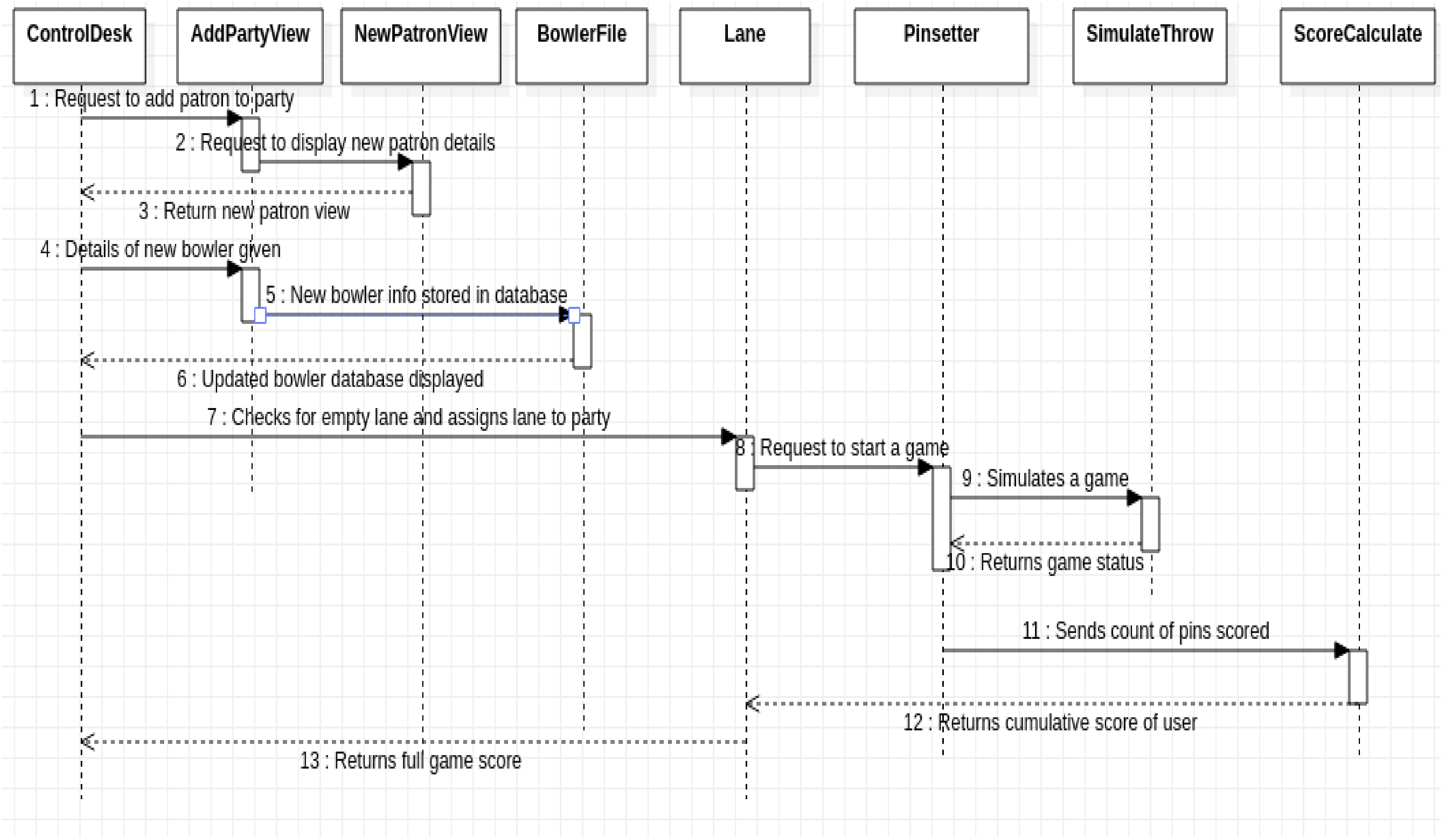
ControlDesk
AddPartyView
NewPatronView
BowlerFile
Lane
Pinsetter
SimulateThrow
ScoreCalculate
1 : Request to add patron to party
2 : Request to display new patron details
3 : Return new patron view
4 : Details of new bowler given
5 : New bowler info stored in database
6 : Updated bowler database displayed
7 : Checks for empty lane and assigns lane to party
8 : Request to start a game
9 : Simulates a game
10 : Returns game status
11 : Sends count of pins scored
12 : Returns cumulative score of user
13 : Returns full game score

DIagram 8: Add new user,start game and calculate score

Responsibilities of classes:-

Class Name	Responsibility
AddPartyView	GUI to add a party
Bowler	Saves all information of a bowler
BowlerFile	Class to store and receive data from bowler database
ControlDesk	Acts like an admin of all the bowling alleys
ControlDeskView	GUI of ControlDesk
Drive	Start simulation of entire program after assigning lanecount and maxpartysize
Lane	Assigns any empty lane to a party and start the game simulation in a lane
LaneStatusView	GUI to display the status of a lane
Party	Saves information of a party of bowlers
PinSetter	Check status of a throw,calculate score and return it to lane
PinSetterView	GUI to check status after a throw
GeneralView	Contains code to add a button and panel to remove redundancy from all view classes
SimulateThrow	Simulates a single ball throw and returns result to PinSetter
ScoreCalculate	Calculate cumulative score after a throw
EndGamePrompt	Prompt to ask user if he wants to play another game
ScoreReport	Sends score report to an user via email
EndGameReport	Receives score report of user,prints it and hands it to user

Analysis of Original Design:-

Weaknesses:-

- **Redundant Code : -** The original code contained a lot of code which was either not used in the actual design or commented out. These unnecessary lines of code, if not removed, might have confused any future developer about the flow of the code. So the redundant or dead codes found in classes like ControlDesk,Lane,PinSetter,etc were all removed.
- **Code Repetitions : -** Code repetitions were found in many classes. Many code modules or tasks were being performed multiple times which was reducing the performance and also the cleanliness. One of the major changes that we did to reduce code repetition was to introduce a new class called GeneralView which contained functions to add a button and panel.Using this single class,all the other view classes can add buttons and panels in their own views instead of each class implementing these functions seperately.
- **Unnecessary comments : -** Comments are added to a code to help the reader understand the logic and purpose of the code.Here in the original code design there was a lack of to the point well documented comments which may confuse the reader.
- **Large classes and functions : -** In the original code,few major classes were performing the bulk of the tasks of the project. A single class was performing many functionalities.To reduce this we broke down large classes into smaller classes with a specific functionality.For example:-PinSetter class was broken down into SimulateThrow which only simulated a ball throw and ScoreCalculate class which only calculated the score of an user.

Strengths:-

- Presence of well defined design patterns while designing subtasks of the project.
- Code functionalities were well defined and proper variable and function names were chosen to aid the code reader.
- The overall project in general had a low coupling metric which is an indication of a good design.Coupling refers to the dependencies between the various classes.

Fidelity to the Design Document:-

The fidelity of the original design documentation was mid fidelity range.Design fidelity refers to the level of details and functionality built into a prototype.Mid fidelity range means that the ideas and logic of the design has already been generated and the reduction phase needs to start which means that we need to discard the ideas not working and pick the best one.

Design Patterns:-

- Observer Pattern : The event handling done in the system on button click is a good example of observer pattern. Here we wait on thread for an event like a button click by the user, and notify the corresponding event-handler which carries out a task corresponding to the given button click.
- Adapter Pattern : Adapter pattern is a structural design pattern that works as a bridge between two incompatible interfaces. In the given system ControlDesk acts as an Adapter by joining Bowlers, Party and Queue subsystems.

- Singleton Pattern : A software design pattern that restricts the instantiation of a class to one "single" instance. This is clearly observed in the drive class, which acts as the main function in this program, and is instantiated only once in its lifetime.