Bowling Game Design Document

Team Members:
- Medha Vempati (2018101063)
- Sai Tanmay Reddy Chakkera (2018101054)
- Balivada Sai Geeth Yaswanth (2018101126)

Date of Submission: April 9th 2020

About The Product

This product is a virtual bowling game. The game allows the user to add players and create parties(groups) of players to play together. Once the game ends, the user can choose to play again with the same party, or exit and make new parties. Prominent features of this game include:

1) New parties are lined up in the party queue and when a lane opens up, the lane is assigned to the next party from the queue.
2) User can also view the pin positions, which are reset after each player's turn.
3) User can view the activity in a lane to see the players respective scores for each round.
4) Once the game ends, the user can request a report consisting of the player's scores.

Initial Structure

After extensive analysis of the initial code base by our team, we concluded that the original code base depicted several standard design patterns. The **builder design pattern** is clearly seen to be implemented in this structure. Classes such as LaneEvent, ControlDeskEvent, Pinsetter etc have interfaces which facilitate the actual construction of their respective classes.

The **observer design pattern** is also evident from the "subscription" based code style, certain events and processes are all reflected in all the objects and instances which are "subscribed" to them.

Another design pattern which seemed to have been followed in the original code design is the **adapter design pattern**. This pattern is especially evident from the interaction between SCOREHISTORY.DA and the PrintableText Class.

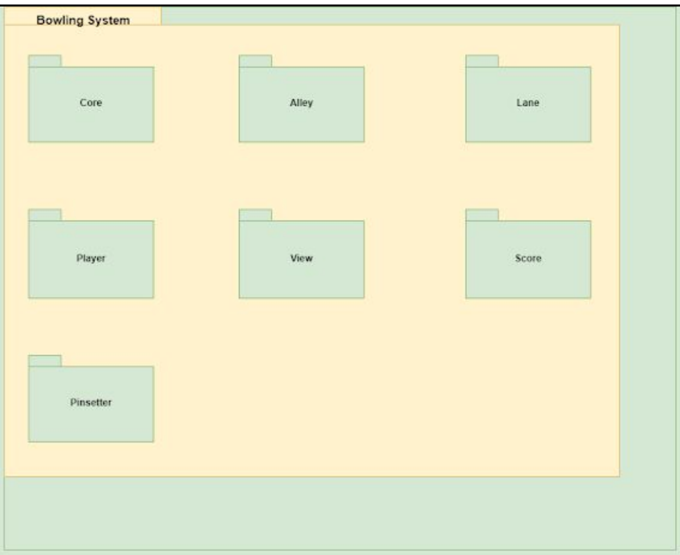
It was also concluded that the code was already fairly modular, but had multiple sections of code which were either redundant and repetitive, or too long and could've been broken down into smaller, more focused parts. These code smells are specified below:

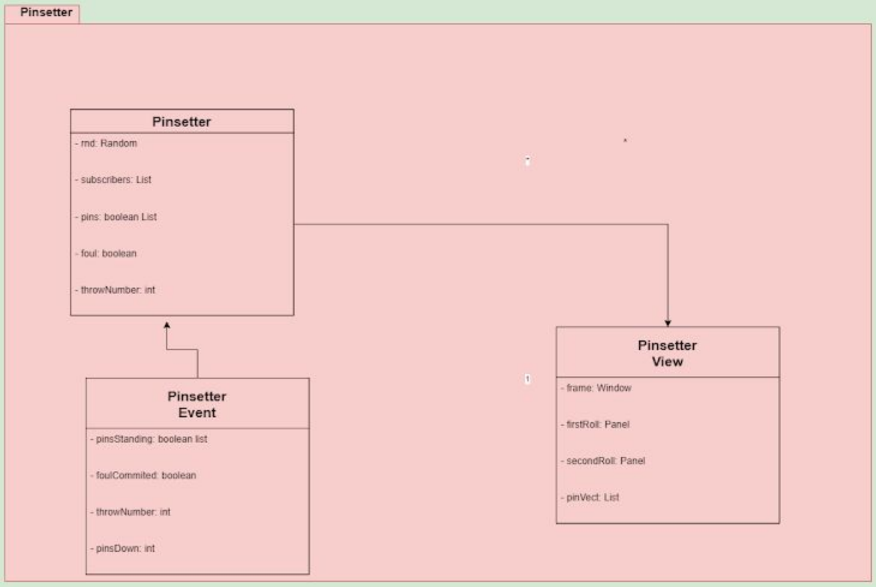
1) The registerPatron() method, although more relevant to the BowlerFile class, is placed in the ControlDesk class.
2) The publish() and subscribe() methods were repeated in the ControlDesk, Lane and Pinsetter classes. This repetition is redundant.
3) The button creation code is repetitive and hence redundant in all the various views of the game.
4) The run() method in the Lane class, has too many if-else statements, making the code harder to comprehend.

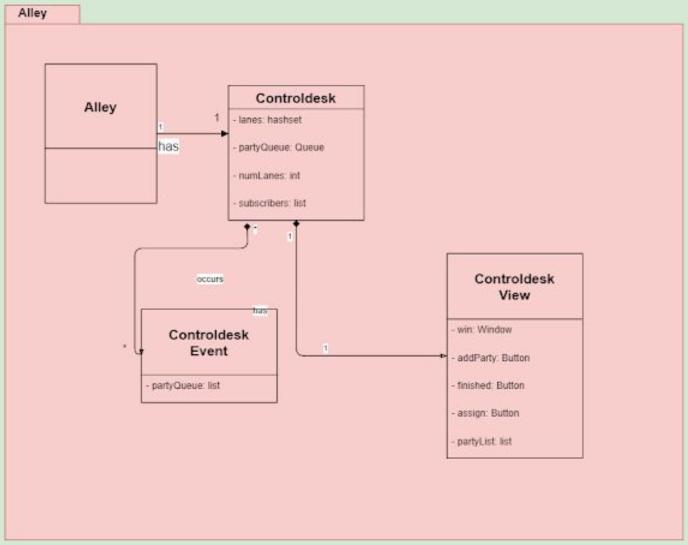
Our team also did a thorough metrics analysis of the original code base. The metrics helped identify the level of complexity of each class in the code base, along with the level of interdependencies between classes.

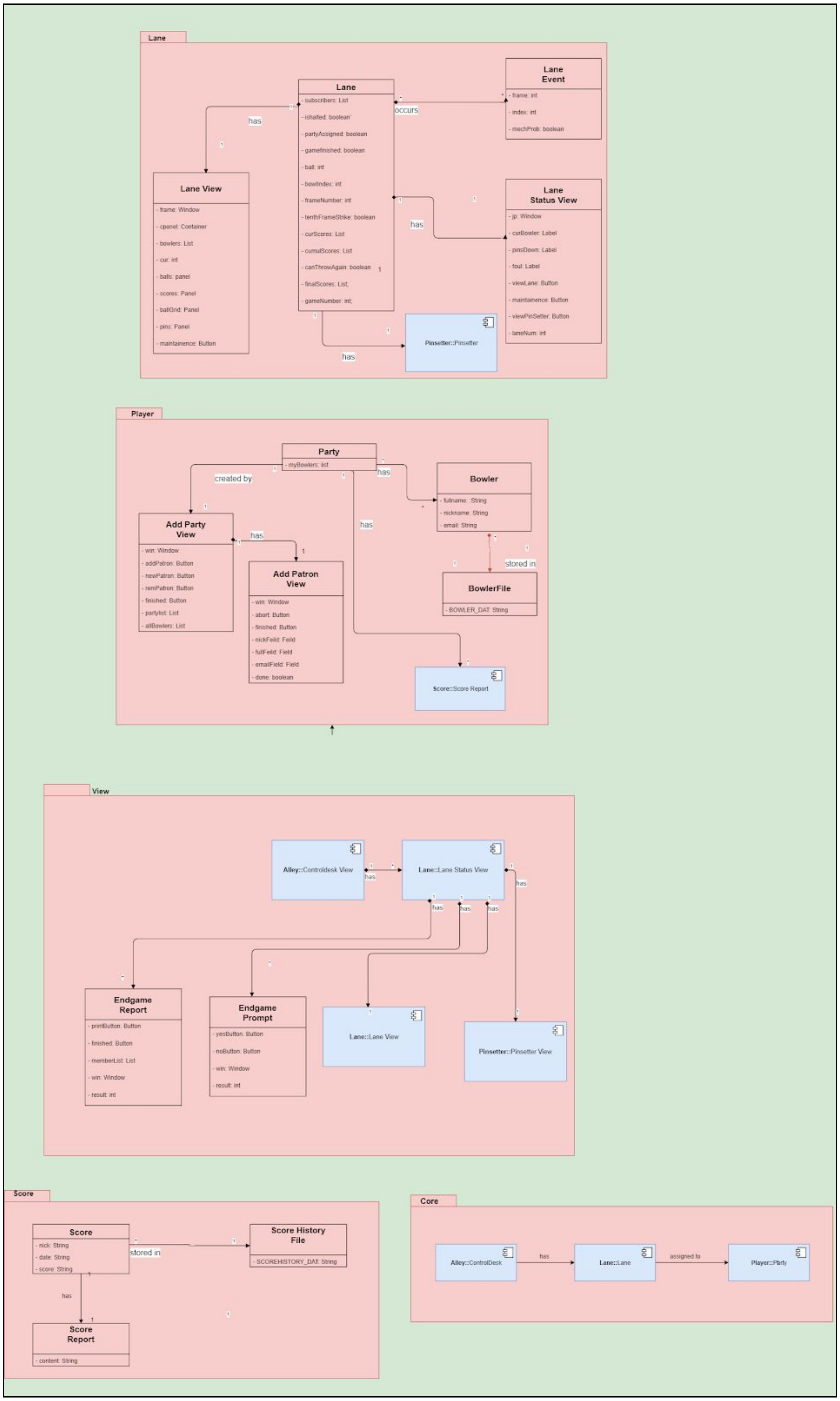
We used these metrics to identify areas of high complexity, and make appropriate changes to reduce the complexity.

The various classes are interdependent on each other as portrayed in the **UML Class Diagram** below:

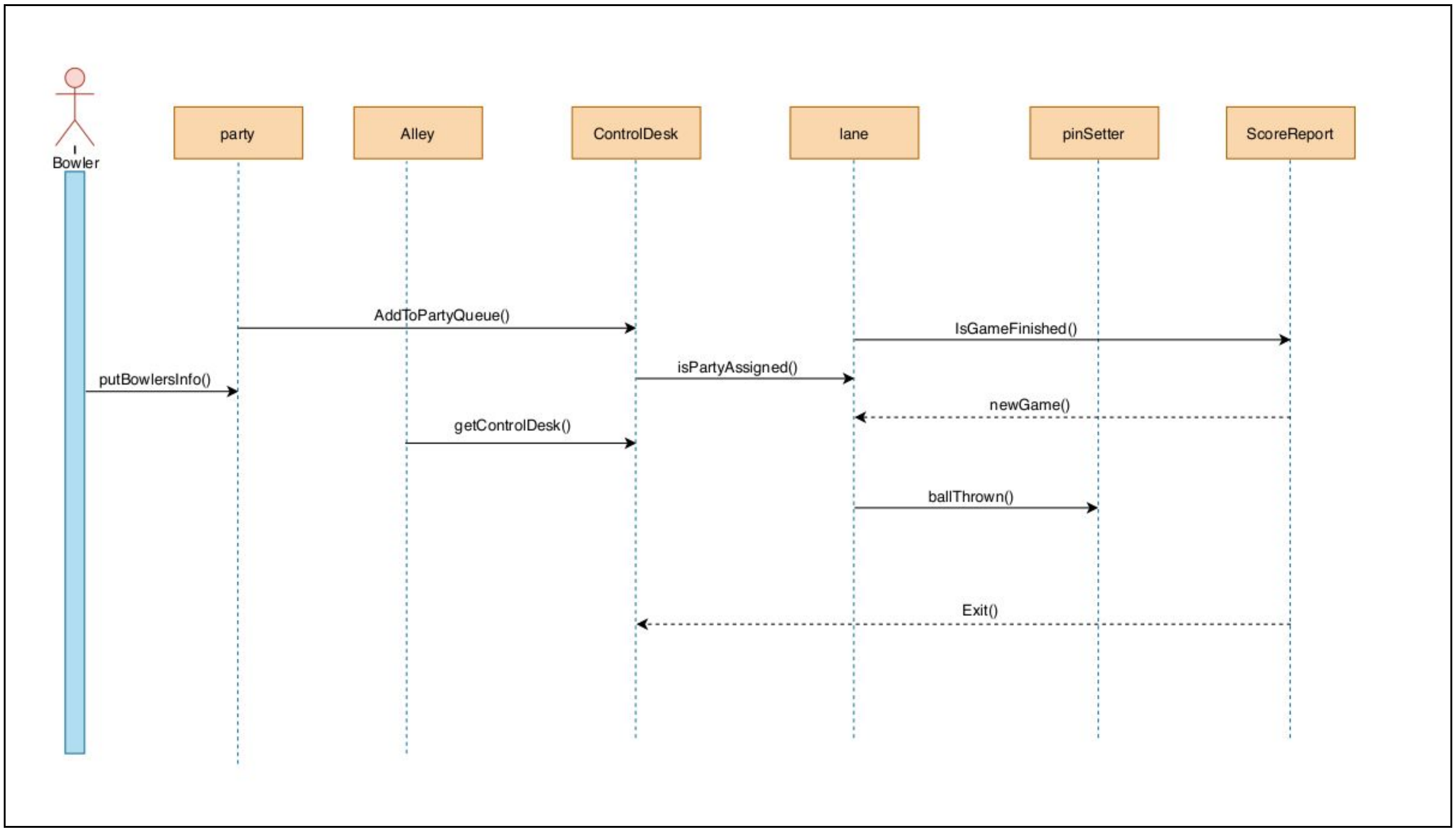

Bowling System
Core
Alley
Lane
Player
View
Score
Pinsetter


Pinsetter
Pinsetter
- rnd: Random
- subscribers: List
- pins: boolean List
- foul: boolean
- throwNumber: int
Pinsetter Event
- pinsStanding: boolean list
- foulCommited: boolean
- throwNumber: int
- pinsDown: int
Pinsetter View
- frame: Window
- firstRoll: Panel
- secondRoll: Panel
- pinVect: List
*
*
1


Alley
Alley
has
1
1
Controldesk
- lanes: hashset
- partyQueue: Queue
- numLanes: int
- subscribers: list
*
1
occurs
has
*
Controldesk Event
- partyQueue: list
1
Controldesk View
- win: Window
- addParty: Button
- finished: Button
- assign: Button
- partyList: list


Lane
Lane
- subscribers: List
- ishalted: boolean'
- partyAssigned: boolean
- gamefinished: boolean
- ball: int
- bowlIndex: int
- frameNumber: int
- tenthFrameStrike: boolean
- curScores: List
- cumulScores: List
- canThrowAgain: boolean
- finalScores: List;
- gameNumber: int;
Lane Event
- frame: int
- index: int
- mechProb: boolean
occurs
has
Lane View
- frame: Window
- cpanel: Container
- bowlers: List
- cur: int
- balls: panel
- scores: Panel
- ballGrid: Panel
- pins: Panel
- maintainence: Button
Lane Status View
- jp: Window
- curBowler: Label
- pinsDown: Label
- foul: Label
- viewLane: Button
- maintainence: Button
- viewPinSetter: Button
- laneNum: int
Pinsetter::Pinsetter
Player
Party
- myBowlers: list
created by
has
Bowler
- fullname: :String
- nickname: String
- email: String
stored in
BowlerFile
- BOWLER_DAT: String
Add Party View
- win: Window
- addPatron: Button
- newPatron: Button
- remPatron: Button
- finished: Button
- partylist: List
- allBowlers: List
Add Patron View
- win: Window
- abort: Button
- finished: Button
- nickFeild: Feild
- fullFeild: Field
- emailField: Field
- done: boolean
Score::Score Report
View
Alley::Controldesk View
Lane::Lane Status View
Endgame Report
- printButton: Button
- finished: Button
- memberList: List
- win: Window
- result: int
Endgame Prompt
- yesButton: Button
- noButton: Button
- win: Window
- result: int
Lane::Lane View
Pinsetter::Pinsetter View
Score
Score
- nick: String
- date: String
- score: String
Score History File
- SCOREHISTORY_DAT: String
Score Report
- content: String
Core
Alley::ControlDesk
Lane::Lane
assigned to
Player::Pbrty

The flow of the initial code structure is visually represented by the **UML Sequence Diagram**:



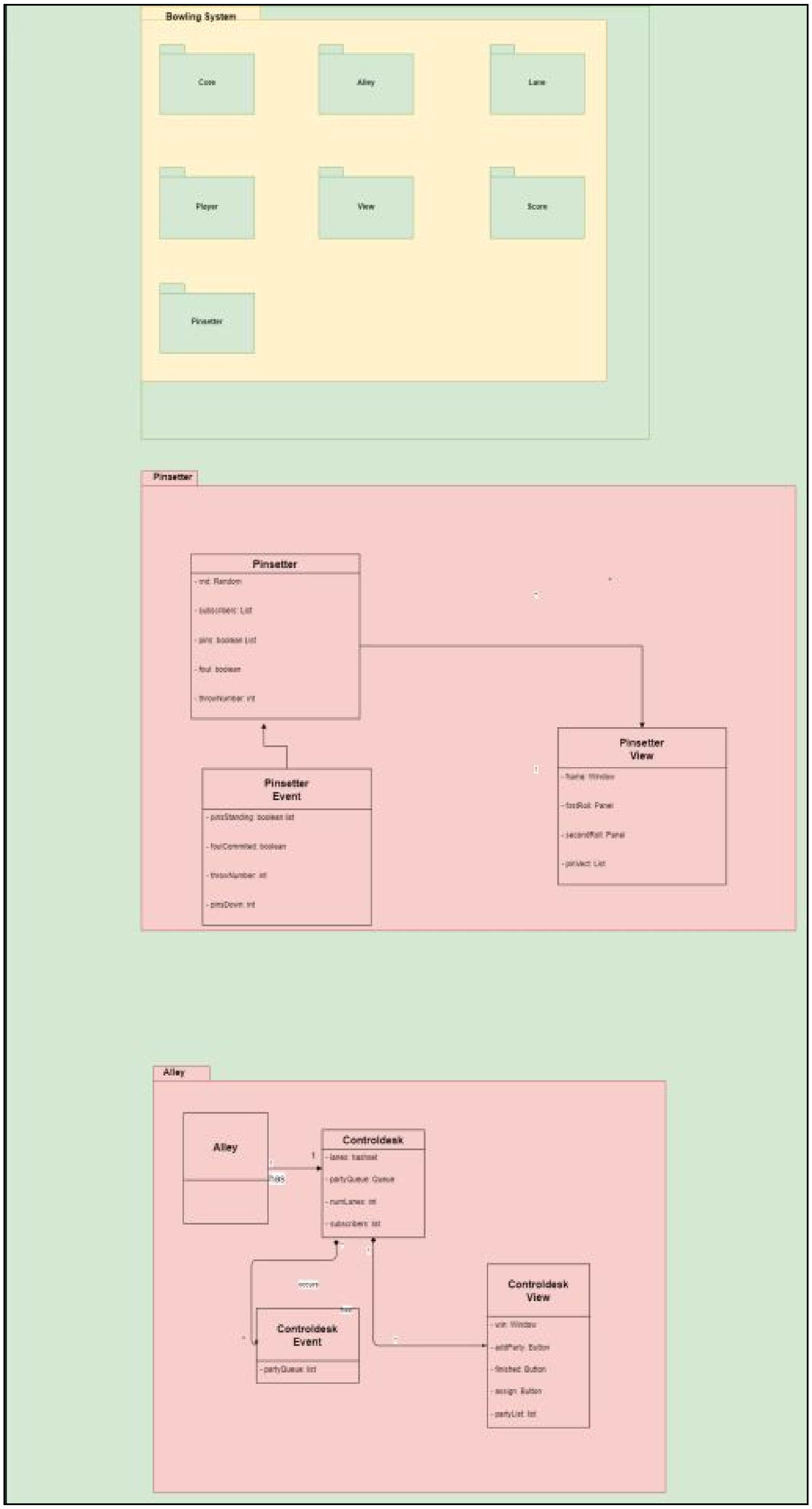
Post-Refactoring

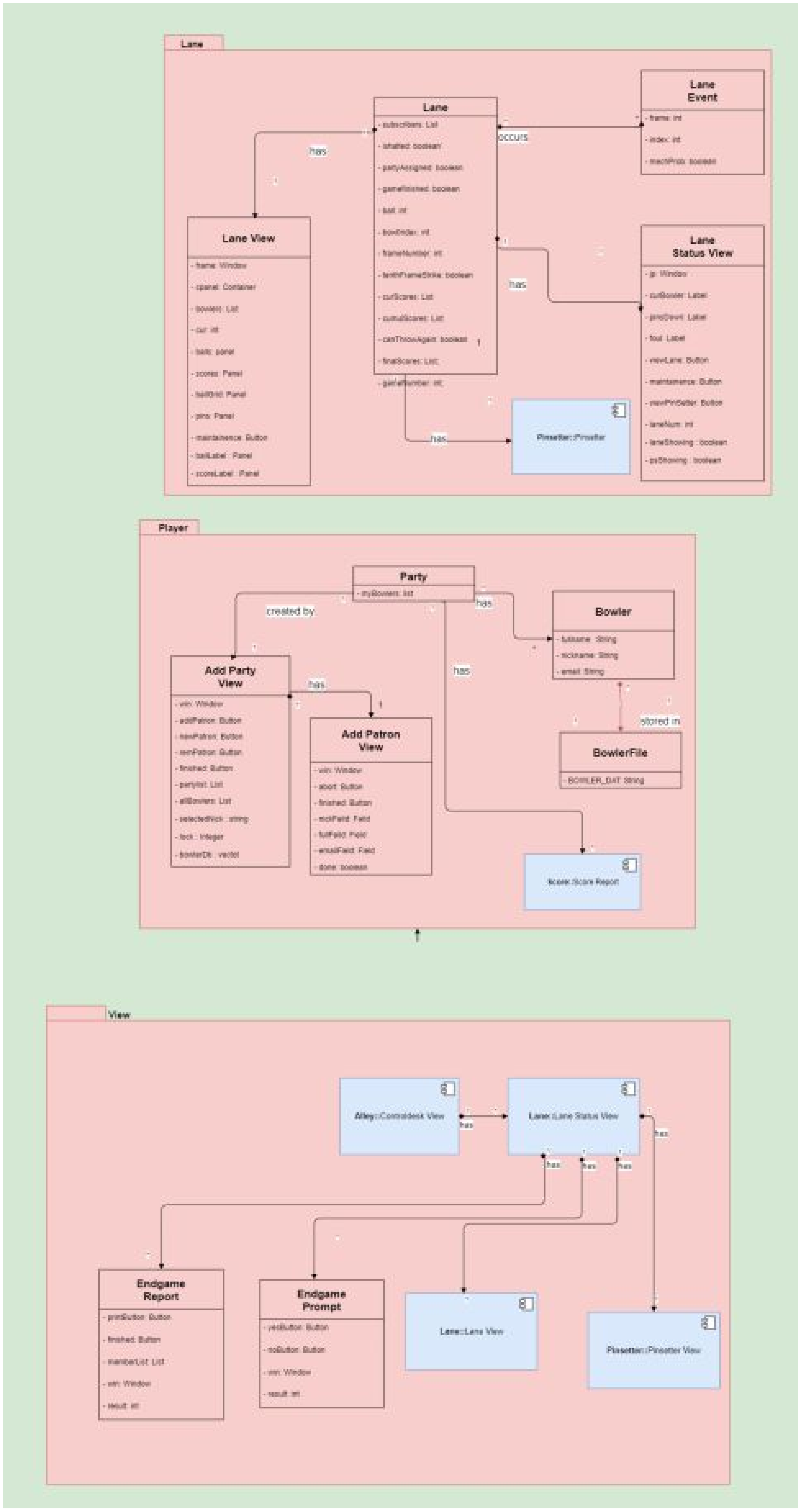
In order to fix these code smells and improve the code, the following changes were made:

1) The registerPatron() method has been moved to BowlerFile class as its more closely related to this class than the previous one.
2) The publish() and subscribe() methods were removed from Lane, Pinsetter and ControlDesk, and put in a new class named PubsAndSubs to avoid repetition. This is done to ensure better separation of concerns.
3) The button creation portion of the code was extracted into a new makeButtonWithPanel() method in each Class to increase modularity.
4) The many if-else statements in the run() method in the Lane class were removed and put into separate functions: playGame() and afterGameEnds()

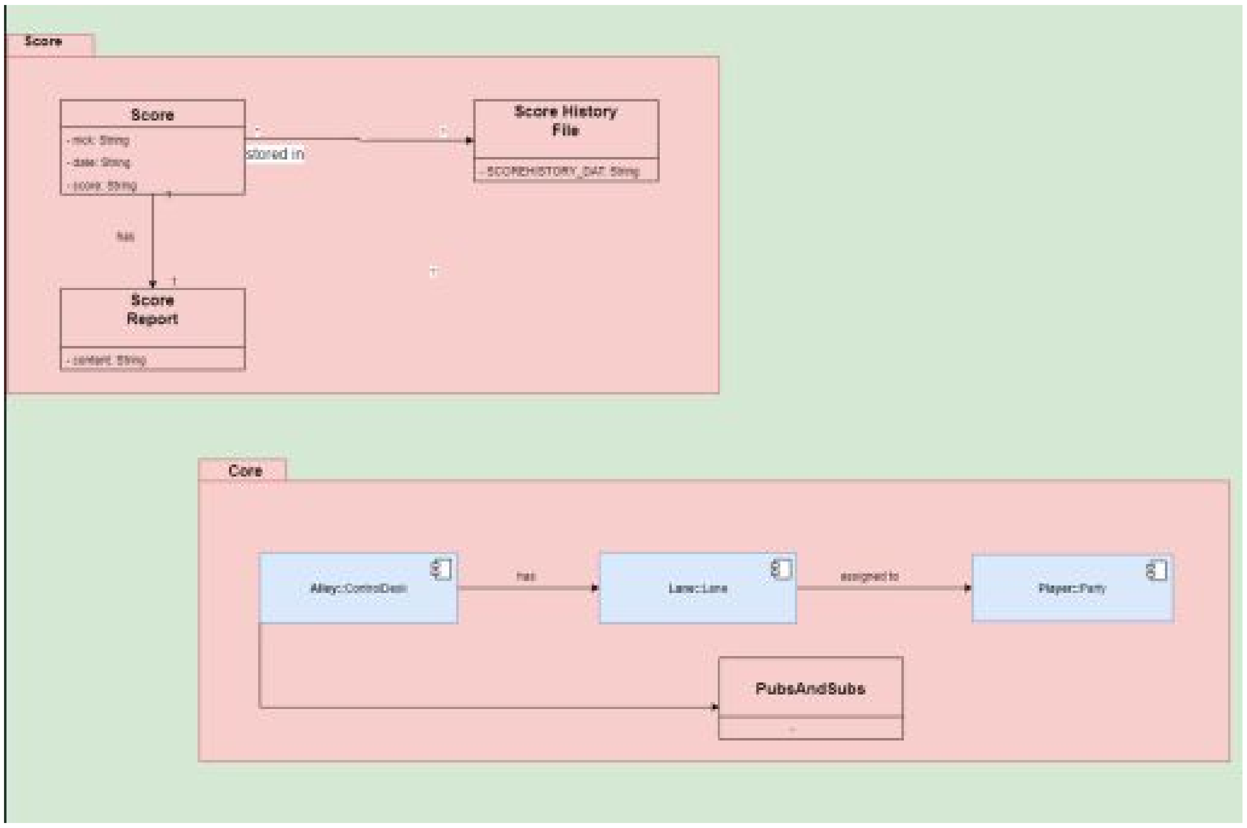
The code refactoring reduced the complexity metrics in multiple classes due to break down of large methods. The inter-dependency metrics remained roughly the same. This showed that the refactoring helped reduce complexity of the code.

After these changes were made, the inter class dependencies and the class structures themselves changed as shown in the **UML Class Diagram** below:


Bowling System
Core
Alley
Lane
Player
View
Score
Pinsetter
Pinsetter
Pinsetter
- rnd: Random
- subscribers: List
- pins: boolean List
- foul: boolean
- throwNumber: int
Pinsetter Event
- pinsStanding: boolean list
- foulCommited: boolean
- throwNumber: int
- pinsDown: int
Pinsetter View
- frame: Window
- firstRoll: Panel
- secondRoll: Panel
- pinVect: List
Alley
Alley
1
has
1
Controldesk
- lanes: hashset
- partyQueue: Queue
- numLanes: int
- subscribers: list
occurs
Controldesk Event
- partyQueue: list
Controldesk View
- win: Window
- addParty: Button
- finished: Button
- assign: Button
- partyList: list



Lane
Lane
- subscribers: List
- halted: boolean
- partyAssigned: boolean
- gameFinished: boolean
- ball: int
- bowlIndex: int
- frameNumber: int
- tenthFrameStrike: boolean
- curScores: List
- cumulScores: List
- canThrowAgain: boolean
- finalScores: List
- gameNumber: int
Lane Event
- frame: int
- index: int
- mechProb: boolean
Lane View
- frame: Window
- cpanel: Container
- bowlers: List
- cur: int
- balls: panel
- scores: Panel
- ballGrid: Panel
- pins: Panel
- maintainence: Button
- ballLabel : Panel
- scoreLabel : Panel
Lane Status View
- jp: Window
- curBowler: Label
- pinsDown: Label
- foul: Label
- viewLane: Button
- maintenance: Button
- viewPinSetter: Button
- laneNum: int
- laneShowing : boolean
- psShowing : boolean
Pinsetter::Pinsetter
has
occurs
has
has
1
1
Player
Party
- myBowlers: list
created by
has
has
has
stored in
Bowler
- fullname : String
- nickname: String
- email: String
BowlerFile
- BOWLER_DAT: String
Add Party View
- win: Window
- addPatron: Button
- newPatron: Button
- remPatron: Button
- finished: Button
- partylist: List
- allBowlers: List
- selectedNick : string
- lock : Integer
- bowlerDb : vector
Add Patron View
- win: Window
- abort: Button
- finished: Button
- nickField: Field
- fullField: Field
- emailField: Field
- done: boolean
1
Score::Score Report
View
Alley::Controldesk View
Lane::Lane Status View
has
has
has
has
has
Endgame Report
- printButton: Button
- finished: Button
- memberList: List
- win: Window
- result: int
Endgame Prompt
- yesButton: Button
- noButton: Button
- win: Window
- result: int
Lane::Lane View
Pinsetter::Pinsetter View


Score
Score
- nick: String
- date: String
- score: String
stored in
Score History File
- SCOREHISTORY_DAT: String
has
1
Score Report
- content: String
Core
Alley::ControlDesk
has
Lane::Lane
assigned to
Player::Party
PubsAndSubs

Class Responsibilities Table

The following table describes the core functionality and purpose of each of the major classes included in the code base

CLASS	RESPONSIBILITY
AddPartyView	Creates a pop-up window which allows the user to add a new group(party) or players to the queue. The user can also add a new bowler to the members list.
Alley	Initialises the ControlDesk with a given number of lanes
Bowler	A single instance of this class holds information(Nickname, Full name, Email) about a single member of the member list.
BowlerFile	Allows us to access and update the members list. We can add new bowlers and update old information.
ControlDesk	It carries out the functionalities in the back that ControlDeskView and AddPartyView offer
ControlDeskEvent	Manages the party queue section of the ControlDeskView
ControlDeskView	Provides a view of the current “queue”, assigned lanes, and the scores in each lane. Also allows the user to add a new party and exit the game.
drive	Launches the game by prompting the ControlDeskView allowing the user to set up a party, or choose an existing one. It also sets up the bowling lanes.
EndGamePrompt	Prompts popup window when the game is over and it gives the user a choice to restart the game or finish. It destroys the current party if they choose to finish
EndGameReport	Prompts pop-up window asking the user whether they’d want to print the report or not.
Lane	Keeps track of the bowling lanes and simulates the bowling game. It assigns lanes to parties according to the party queue and computes the scores. Ensures all players and party gets their turn.
LaneEvent	Holds lane attributes like frame number, current bowler, throw number, score etc.

LaneStatusView	This shows us the central panel of the ControlDesk where we can see the lanes, the current bowler in each lane, pins down and etc. It also provides us with an option to see properly the selected lane status and pins status
LaneView	This view shows the pins positions as the game progresses.
NewPatronView	Prompts a window asking the user for information about the new member.
Party	A single instance of this class just holds the information about a single party which includes the bowlers in the party.
Pinsetter	Resets pin positions after each player's turn. It also randomly simulates the throws.
PinsetterView	Opens an interface which shows the dropping of pins so we can see the results
PrintableText	Formats information to be printed in the report when the user requests it.
Queue	An instance of this class contains the party queue for a single instance of the game.
Score	A single instance holds information about a single player's score.
ScoreHistoryFile	Updates the scores in the SCOREHISTORY.DAT file (contains history of scores for each player)
ScoreReport	Generates the report for each player consisting of their scores after the game and if requested, prints/emails it to them.