Bowling Alley

Simulation Refactoring

Team

Gajanan Modi (2019201049)

Utkarsh Upendra (2020900047)

Aditya Khandelwal (20171117)

Antony Martin (20171030)

...

Overview

A bowling establishment is commonly referred to as a bowling alley. A bowling alley is composed of a number of bowling lanes. Bowlers may check in as a group or party so that they will be assigned to the same lane. Each lane can accommodate one to five bowlers. The order in which a party checks in determines the order in which they will bowl.
Once a party has started bowling, the control desk can monitor the number of frames completed by each bowler. Each lane is equipped with a stand-alone scoring station that lists the bowlers' names (in the order they checked in) and a graphic representation of their scores. Bowlers alternate rolling the ball or taking throws according to the scoring rules of bowling referenced in the following section

Initial

Class Responsibility Table

Class	**Responsibility**
AddPartyView	Creates a popup window with the option to add or remove a patron for a party. It also provides us an option to create a patron which will be in the party. Game is started by pressing the Finished button
Alley	Initialises the ControlDesk with a given number of lanes
BowlerFile	Provides us a way to interact with the list of Bowlers in following ways - fetching / updating information of bowlers, adding / removing bowlers
ControlDesk	Represents control desk and carries out functionalities of the program specified by AddPartyView and ControlDeskView

ControlDeskView	Creates view, provides us with an UI for us to add a party or to finish the game. And also provides us with the view of waiting parties and the current lanes assigned.
drive	Starts game by creating Alley
EndGamePrompt	Creates a popup window when the game is over and it provides us the option to restart the game or finish through its GUI interface
EndGameReport	Provides option to print the report or not using a popup window.
Lane	Keeps track of the current lanes and simulates the bowling game. It assigns lanes to parties, computes the scores, designs the functioning of the game(ensures everyone is getting their turn at the right time)
LaneEvent	Holds the values which define a lane like the frame number, current bowler, throw number and all.
LaneStatusView	Creates the view shown in the center of the ControlDesk where things like current bowler in each lane, lanes, pins down and etc are shown. It also provides us with an options to see pins status
LaneView	Creates view for the number of pins down in each throw for each player
NewPatronView	Creates view which lets us input the details about the new patron we are registering
Pinsetter	Simulates the dropping of pins in the lane by updating the states of each pin. Results are randomly generated for each throw
PinsetterView	Creates view which shows the status of pins so we can see what's going on the Lane
Score	Sets the scores for the players in a game
ScoreHistoryFile	Updates the scores in the .DAT file which contains history of scores for each player
ScoreReport	Generates the report for each player at the end of a game and prints/emails it. It includes the current score and previous scores for the given player

Narrative

Weaknesses

Classes weren’t independent, coupling was medium overall. Relatively less reuse of code was observed. Also many methods in classes do multiple tasks and can be split into multiple methods which are much simpler and easy to understand. A few typos were found here and there in variable and method names. Work done by the classes wasn't balanced, for example class Lane did a lot of heavy work whereas some others were just data classes like LaneEvent.

Strength :

The coupling amongst all the classes has been kept low .A strong cohesion amongst related classes exists. The size of the code files has been kept low and optimum. The model classes are highly independent. The control classes communicate between view and model classes as expected.

Fidelity To Design Documentation

● A bowling alley is composed of a number of bowling lanes and parties bowl on these lanes

● Each lane is equipped with a stand-alone scoring station that lists the bowlers’ names and a graphic representation of their scores.

● The pinsetter interface communicates to the scoring station the pins that are left standing after a bowler has completed a throw.

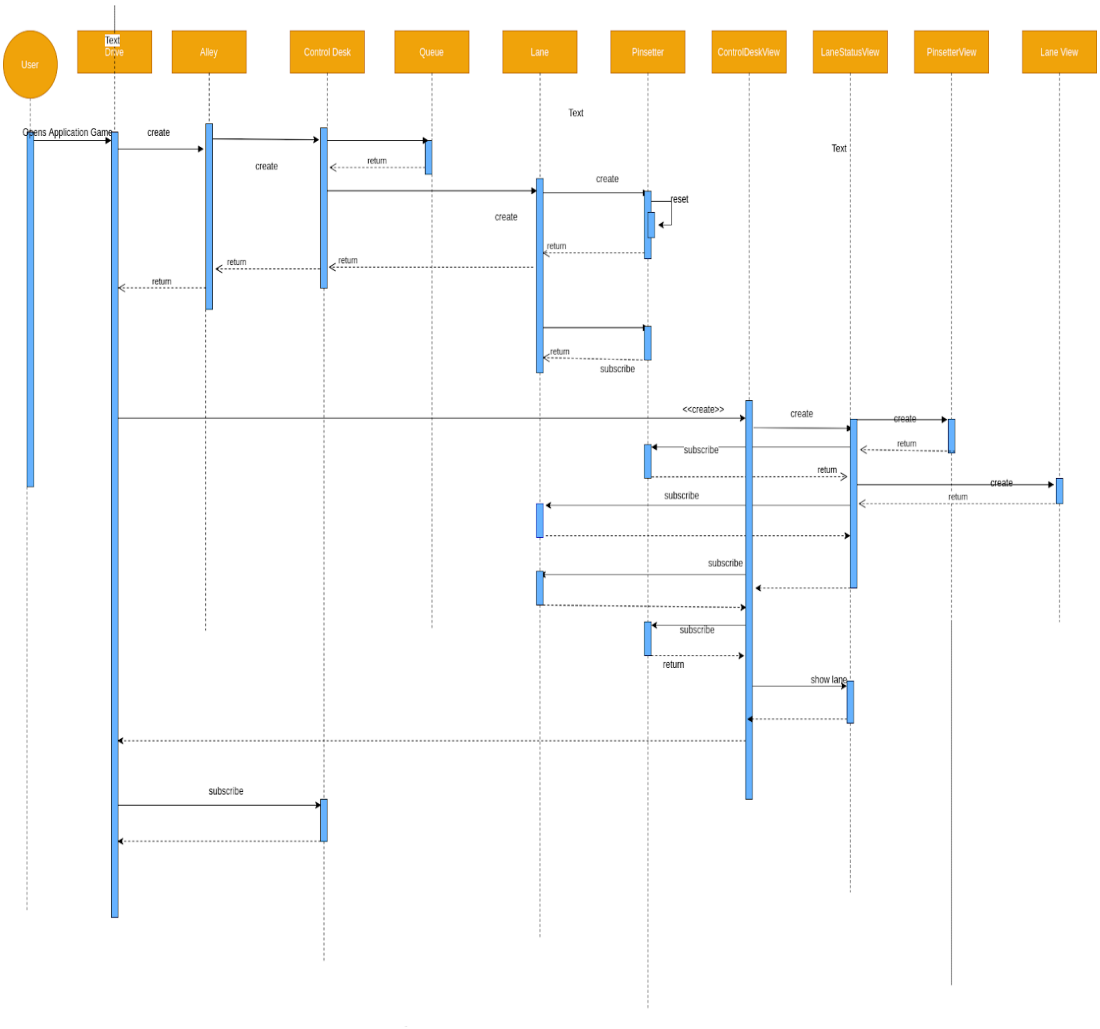
●The control desk operator has the ability to monitor the scores of any active lane.

Code Smells Table

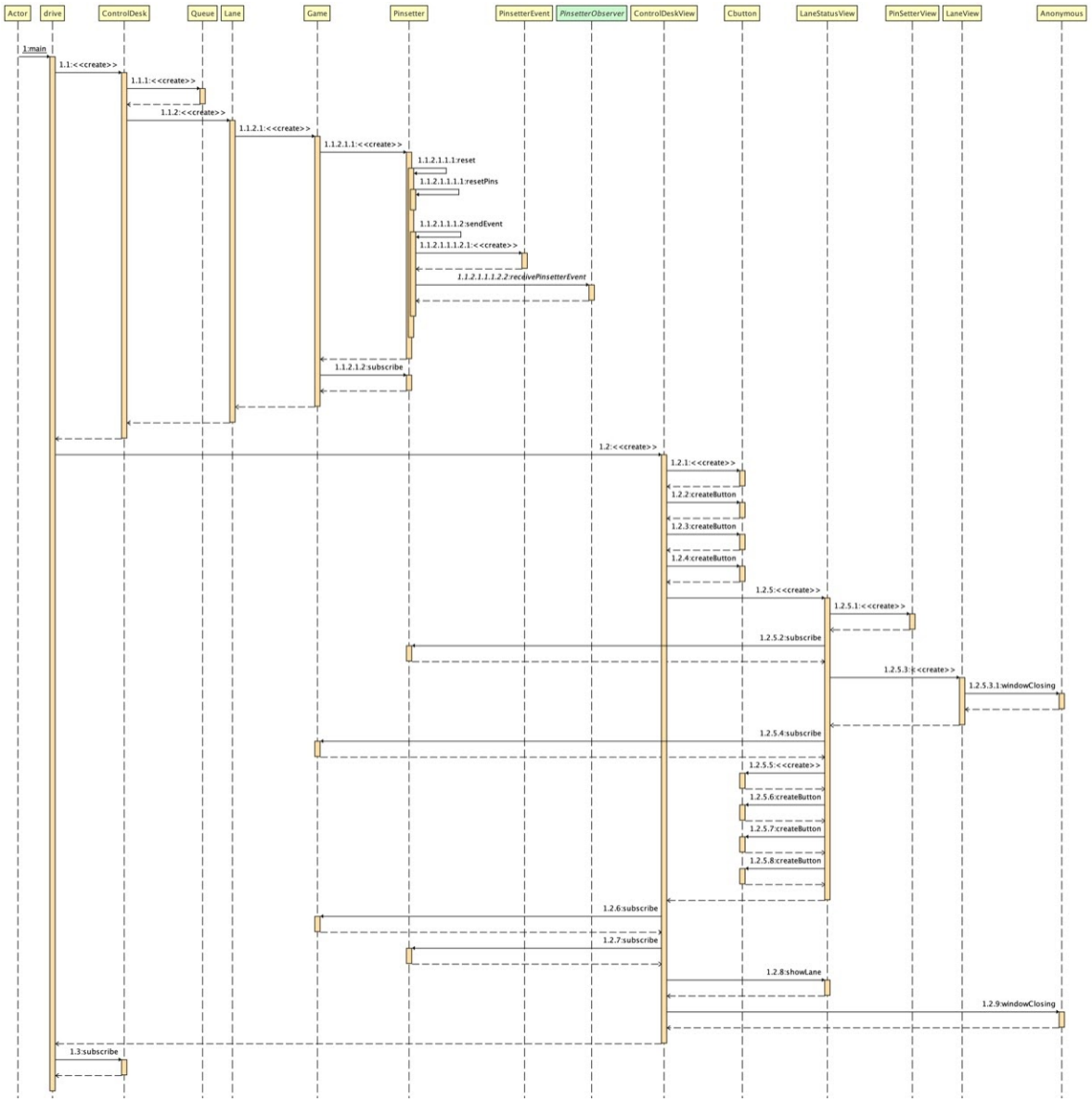
Class File	Code Smell Type	Code Smell
AddPartyView	Code Repetition	Making buttons
AddPartyView	Deprecation	methods show(), hide()
AddPartyView	High Complexity	
ControlDesk	Dead Code	Unused method viewScores()
ControlDeskEvent	Data Class	Only used for storage, and methods are getters only. No methods were operating on data.
ControlDeskEvent	Middleman	ContrlDesk Class was delegating it work to ContrlDeskEvent class
ControlDeskView	Code Repetition	Making buttons
ControlDeskView	Redundancy	assignPanel button not being used
GameEvent	Data Class	Only used for storage, and methods are getters only. No methods were operating on data.
GameEvent	Middleman	Game Class was delegating it work to GamekEvent class
GameEvent	Dead Code	Unused getters in GameEvent Class
Lane	Large Class	Many methods and variables
Lane	High Complexity	getScore(), recievePinsetterEvent(), run() methods have very high complexity due to if-else conditions
Lane	Redundancy	
LaneStatusView	Redundancy and Complexity	A if condition was repeated thrice in actionPerformed()

LaneEvent	Dead Code	Unused getFrame() and getCurScores()
LaneEvent	Data Class	Only used for storage, and methods are getters only. No methods were operating on data.
LaneEvent	Middleman	Lane Class was delegating it work to LaneEvent class
NewPatron	Code Repetition	Making buttons
PinsetterView	Code Repetition	Making buttons
PinSetterEvent	Middleman	PinSetter Class was delegating it work to PinsetterEvent class
PinSetterEvent	Dead Code	Unused getters in GameEvent Class

Initial Sequence Diagram



Final Sequence diagram



Note: all diagrams can be referred from misc folder

Refactoring

Narrative

We have Refactored the initial codebase using various techniques, they are listed below. The decision to take steps was influenced by metrics and also general readability of code. The metrics are listed in the next section where we describe each of the metrics we have used and analyse our results of refactoring by comparison with initial values.

One of the major tasks that was taken up was to refactor “Lane” class, which as per the metrics had low cohesion. Hence, the decision was made to split the Lane class to create a new class called “Game”. To closely resemble real world entities, it made even more sense to host the game related attributes like scores, ball indices and bowlers in the game class itself. This also led to creation of a new interface - “GameObserver” which took up most of the responsibilities of the “LaneObserver” interface. As per the refactored design, the only event that a LaneObserver listens to is that of a mechanical problem, and all the other events related to game are processed by the GameObserver.

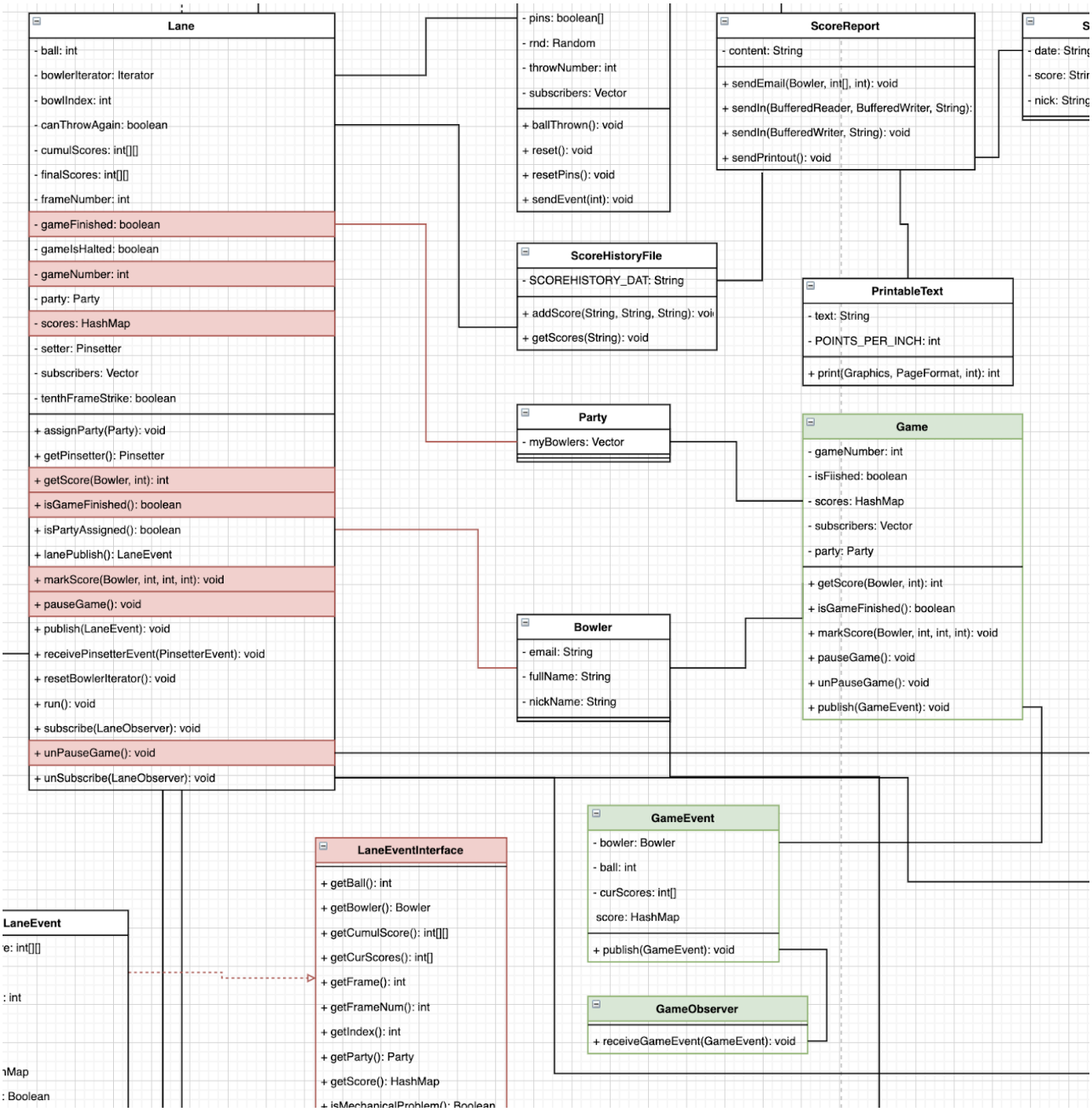
Another task we took up was to change the way the observer pattern was implemented. In the old design, the observables were not notifying the observers with its attributes directly. Another class was being used to store data without performing any operations on it and this class was being used to send data to the observers. This was found to be unnecessary so all these classes(event classes) were removed and the required attributes were passed directly to reduce clutter and simplify the structure.

UML Class Diagram:

The complete UML diagram can be found under the misc directory of the submission zip.

Editable can be found at:

https://app.diagrams.net/#G14p2oypoURta2Myz_uxdQZZxW8j9d_cxq



(*note changes from initial uml diagram are shown with red and green colors)

Comparison of metrics before and after Refactoring

Lines of Codes Metrics:

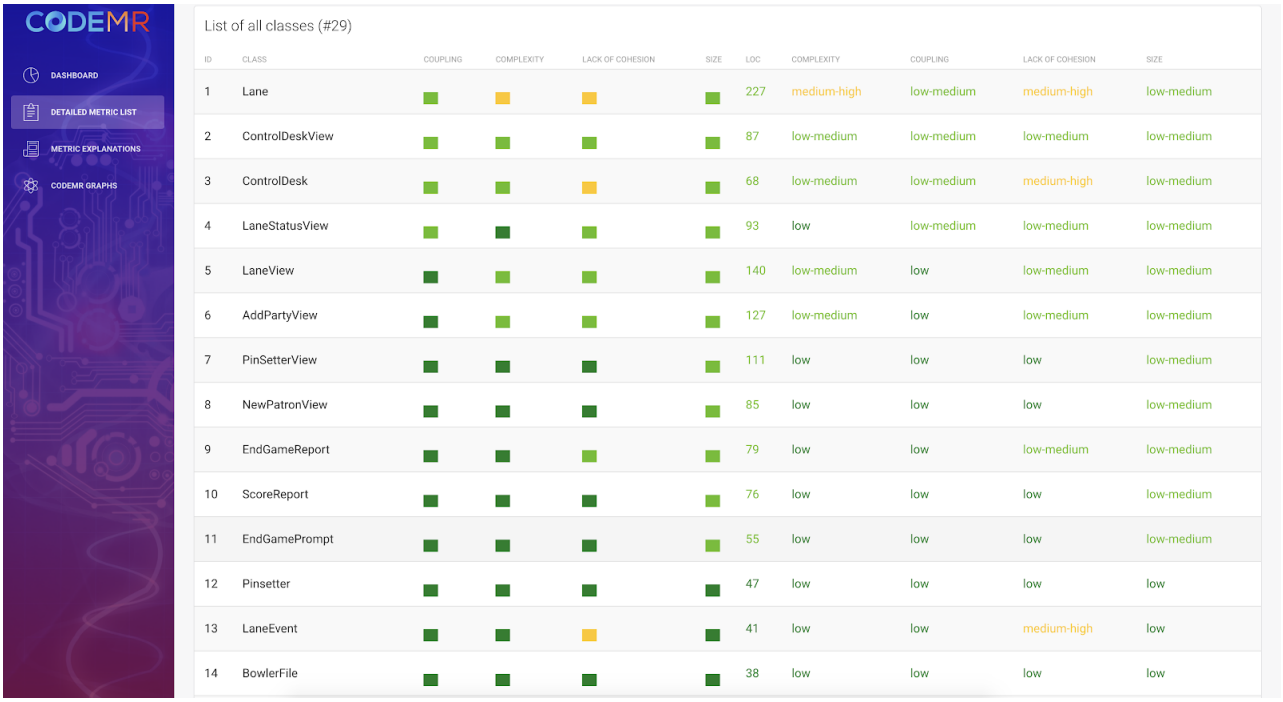
With this value of lane class leads us to split the responsibility of that class we reduces lane class by introducing new game class and removing some redundant code
After that we also check all view classes we get some redundant code in view class and try to remove it and reduce the number of lines
Also we remove dead code and unused methods from the classes improve avg value

	Initial (LOC)	**Refactored (LOC)**
Lane Class	419	85
View.AddPartyView	172	151
...	...	...
Avg (whole code)	90.46	78.81

Complexity Metrics
Here we tried to reduce Operational average complexity by making some changes as mentioned in codesmell table.Though for some classes it might get increase but overall complexity get reduce

	Initial (OCavg)	**Refactored (OCavg)**
Lane Class	4.34	2.57
LaneStatusView	3.40	3
...	...	...
Avg (whole code)	2.13	1.93

Before Refactoring:



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Lane					227	medium-high	low-medium	medium-high	low-medium
2	ControlDeskView					87	low-medium	low-medium	low-medium	low-medium
3	ControlDesk					68	low-medium	low-medium	medium-high	low-medium
4	LaneStatusView					93	low	low-medium	low-medium	low-medium
5	LaneView					140	low-medium	low	low-medium	low-medium
6	AddPartyView					127	low-medium	low	low-medium	low-medium
7	PinSetterView					111	low	low	low	low-medium
8	NewPatronView					85	low	low	low	low-medium
9	EndGameReport					79	low	low	low-medium	low-medium
10	ScoreReport					76	low	low	low	low-medium
11	EndGamePrompt					55	low	low	low	low-medium
12	Pinsetter					47	low	low	low	low
13	LaneEvent					41	low	low	medium-high	low
14	BowlerFile					38	low	low	low	low

15	PinsetterEvent					26	low	low	low	low
16	Bowler					25	low	low	low	low
17	PrintableText					21	low	low	low	low
18	ScoreHistoryFile					20	low	low	low	low
19	Score					16	low	low	low	low
20	Queue					12	low	low	low	low
21	LaneEventInterface					10	low	low	low	low
22	drive					8	low	low	low	low
23	Alley					6	low	low	low	low
24	ControlDeskEvent					6	low	low	low	low
25	Party					6	low	low	low	low
26	PinsetterObserver					2	low	low	low	low
27	ControlDeskObserver					2	low	low	low	low
28	LaneServer					2	low	low	low	low
29	LaneObserver					2	low	low	low	low

After Refactoring:

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Game					206	low-medium	low-medium	low-medium	low-medium
2	ControlDeskView					74	low-medium	low-medium	low-medium	low-medium
3	ControlDesk					68	low-medium	low-medium	medium-high	low-medium
4	Lane					59	low-medium	low-medium	low	low-medium
5	LaneStatusView					80	low	low-medium	medium-high	low-medium
6	LaneView					140	low-medium	low	low-medium	low-medium
7	AddPartyView					107	low-medium	low	low-medium	low-medium
8	PinSetterView					110	low	low	low	low-medium
9	EndGameReport					78	low	low	low-medium	low-medium
10	ScoreReport					76	low	low	low	low-medium
11	NewPatronView					73	low	low	low	low-medium
12	EndGamePrompt					55	low	low	low	low-medium

15	GameEvent					35	low	low	medium-high	low
16	PinsetterEvent					26	low	low	low	low
17	Bowler					25	low	low	low	low
18	PrintableText					21	low	low	low	low
19	ScoreHistoryFile					20	low	low	low	low
20	Score					16	low	low	low	low
21	Cbutton					12	low	low	low	low
22	Queue					12	low	low	low	low
23	drive					7	low	low	low	low
24	Alley					6	low	low	low	low
25	Party					6	low	low	low	low
26	LaneEvent					6	low	low	low	low
27	ControlDeskEvent					6	low	low	low	low