



# Refactoring Bowling Lanes

By Group 5: JT Novitsky, Ethan Ligi, Glenn Todd, Michael Ackerman




## Primary refactoring

method	▼	CogC	ev(G)	iv(G)	v(G)
Lane.getScore(Bowler,int)		111	5	1	38
Lane.run()		55	1	14	19
LaneView.receiveLaneEvent(LaneEvent)		36	1	17	19
Lane.receivePinsetterEvent(PinsetterEvent)		25	1	9	12
LaneStatusView.actionPerformed(ActionEvent)		17	1	11	11


We decided to focus on the complexity metrics of the code, focusing on the five most complex methods in the project


## Secondary Refactoring

Hot spot area in code base	Evidence found
JTA/JTM	0 test assertions and 0 test methods
JLOC (lines of javadoc)	Total:330 Average: 2.6
Implement better packaging 	Only one package for every class used for the entire project
Naming of classes	"drive.java"
Magic numbers	for loops and if statements equalling integers with no context
God Class	Lane.java

Other things we noticed about the code during our initial analysis

- Magic numbers
- Poor testing
- Poor commenting

class 	JTA	JTM
<b>Total</b>	<b>0</b>	<b>0</b>
Average	0.00	0.00

class 	Jf	JLOC	Jm
<b>Total</b>		<b>371</b>	
Average	6.43%	15.46	44.53%

---

# What We Refactored

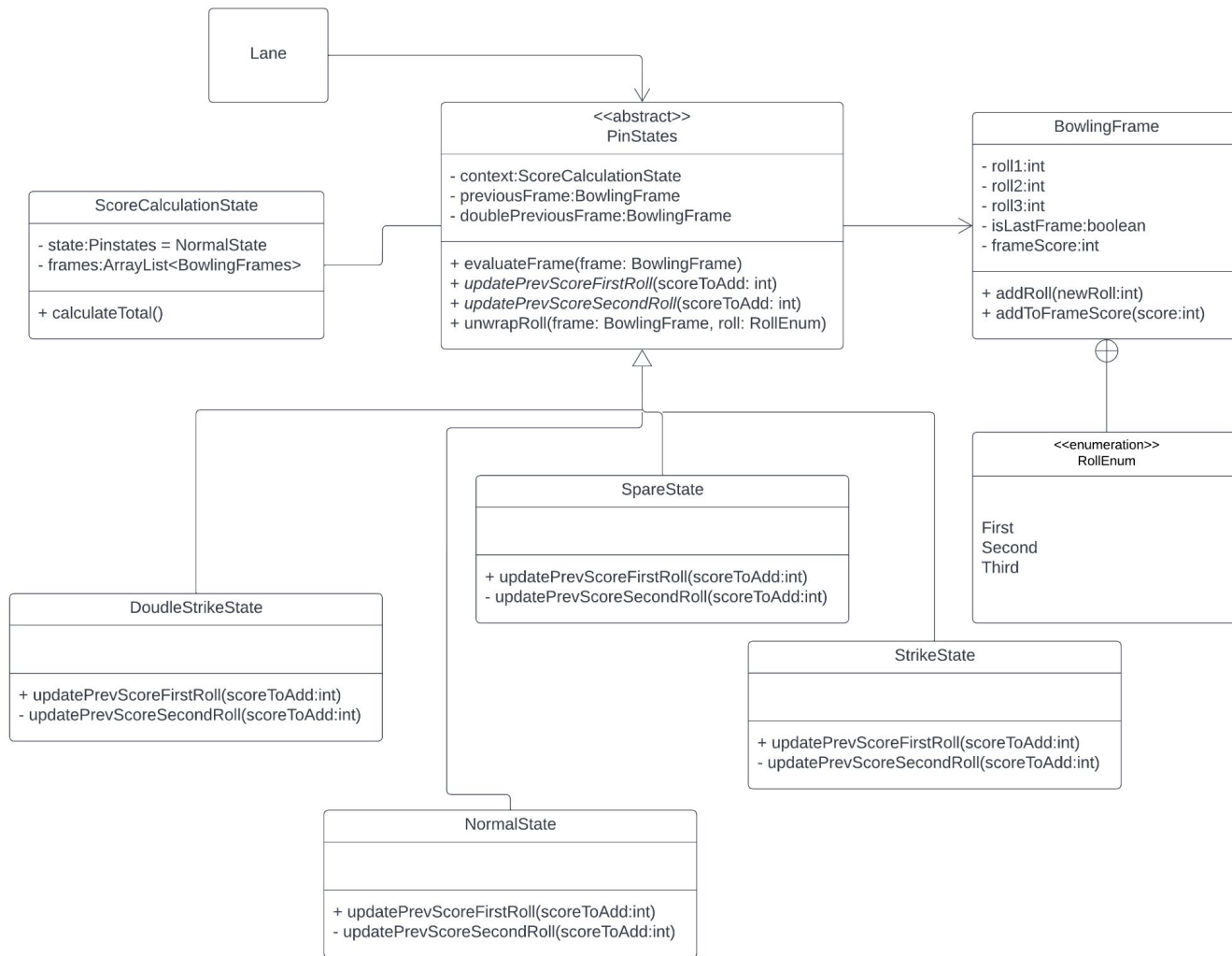


## Lane.getScore()

### State Pattern

- Depending on the state of the throw, the score would be calculated differently
- Strike, Double Strike (10th frame), Spare, and Normal

method	▼	CogC	ev(G)	iv(G)	v(G)
Lane.getScore(Bowler,int)		1	1	2	2



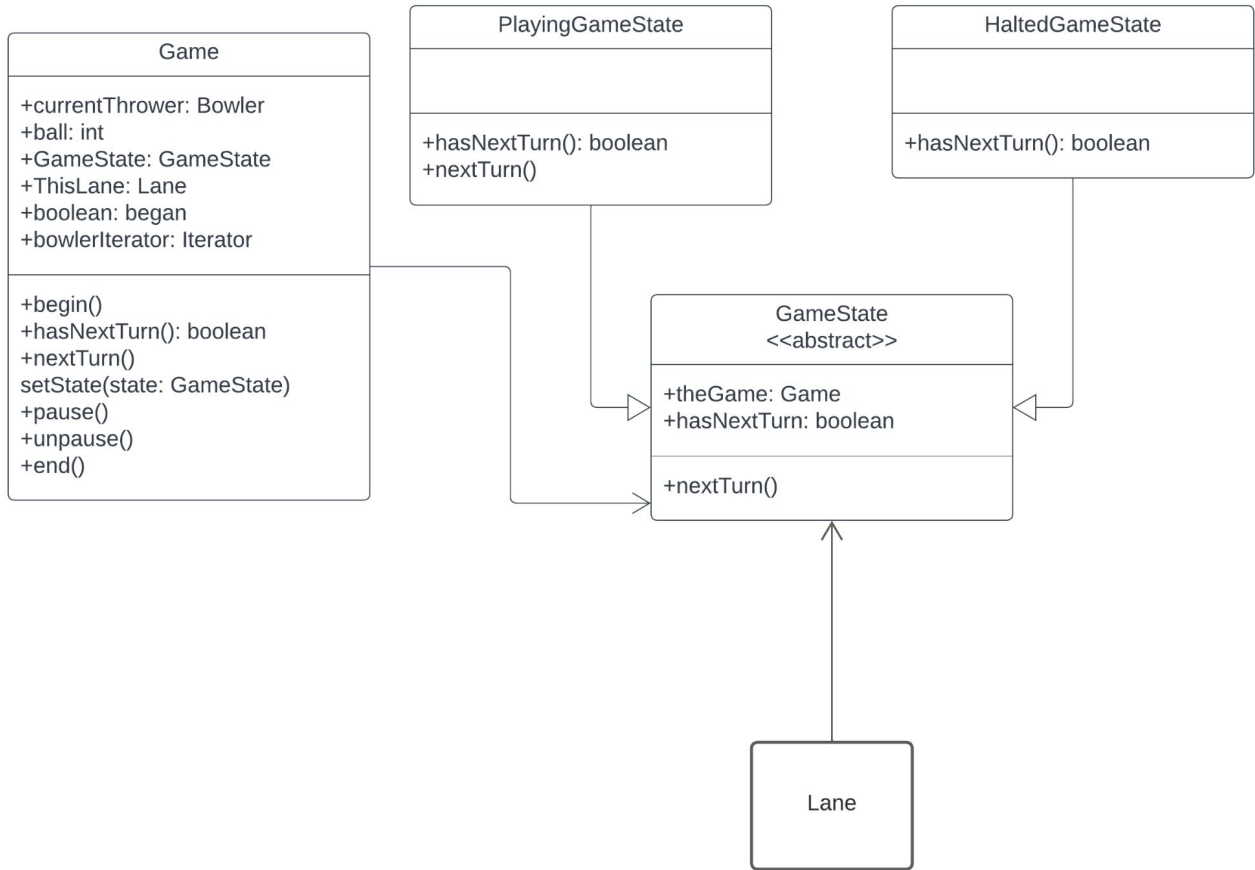


# Lane.run()

## State Pattern

- Depending on the number of bowlers left, current frame, and play again option: bowl accordingly
- Ran into bugs prohibiting completion

method	▼	CogC	ev(G)	iv(G)	v(G)
Lane.run()		17	1	5	7





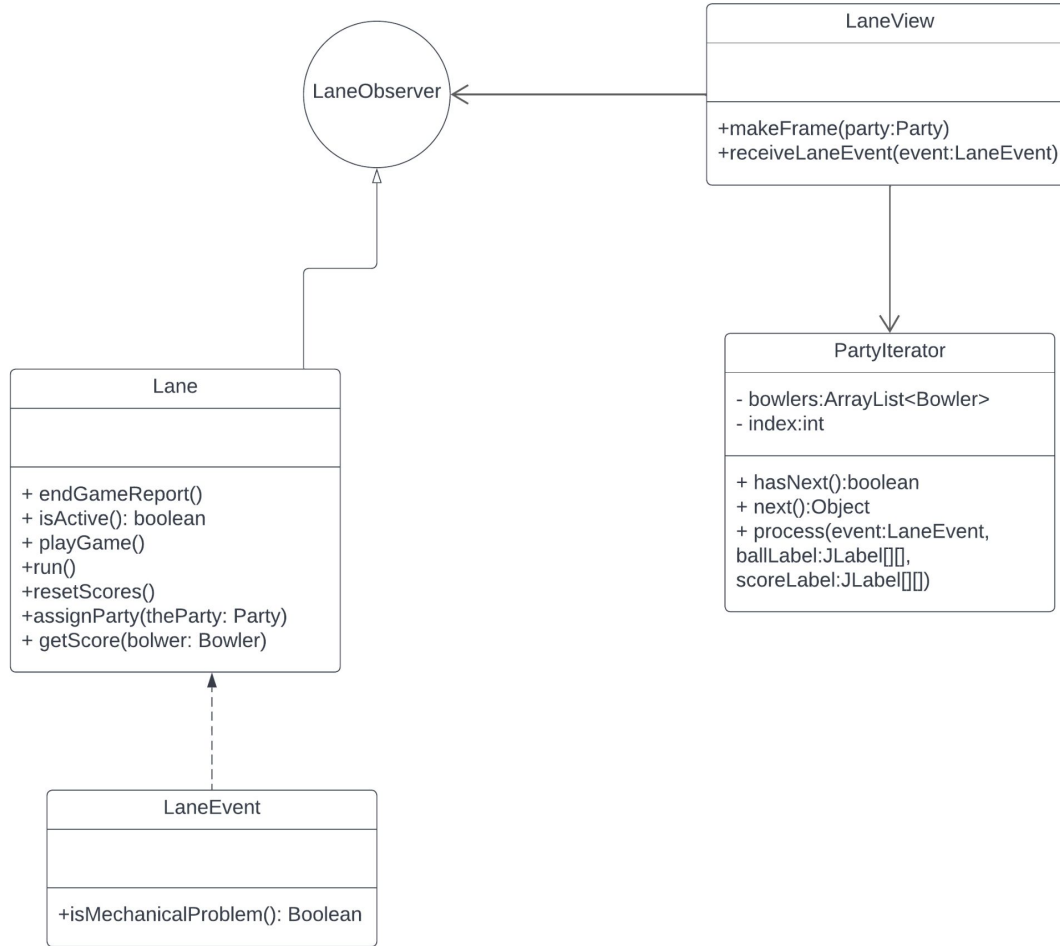


# LaneView.ReceiveLaneEvent()

## Observer Pattern

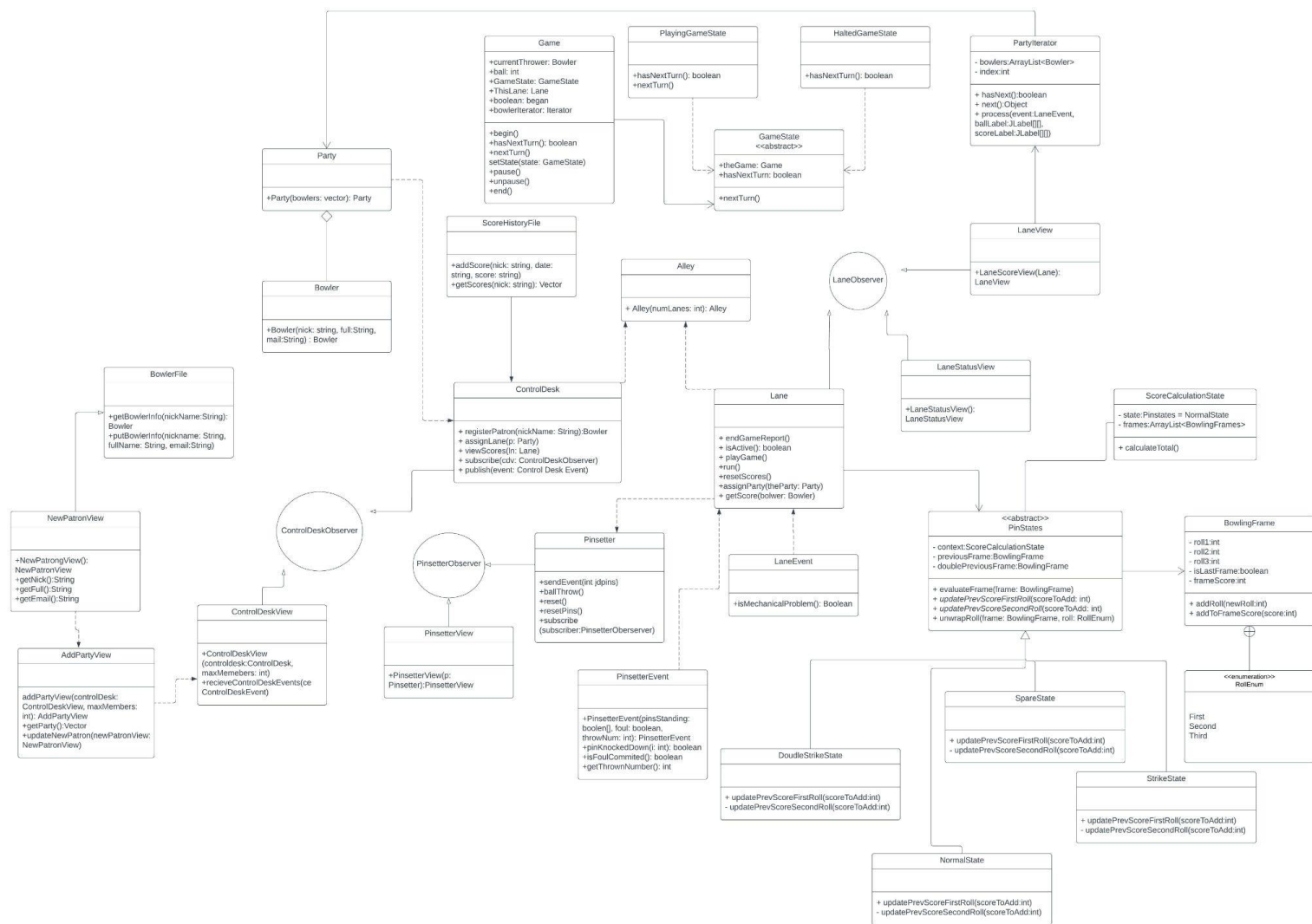
- Took existing observer pattern and refactored to reduce code complexity

method	▼	CogC	ev(G)	iv(G)	v(G)
Lane.receivePinsetterEvent(PinsetterEvent)		25	1	9	12
LaneView.receiveLaneEvent(LaneEvent)		9	1	6	7





**New Class Diagram Below(Next Slide)**



---

# Sequence Diagrams

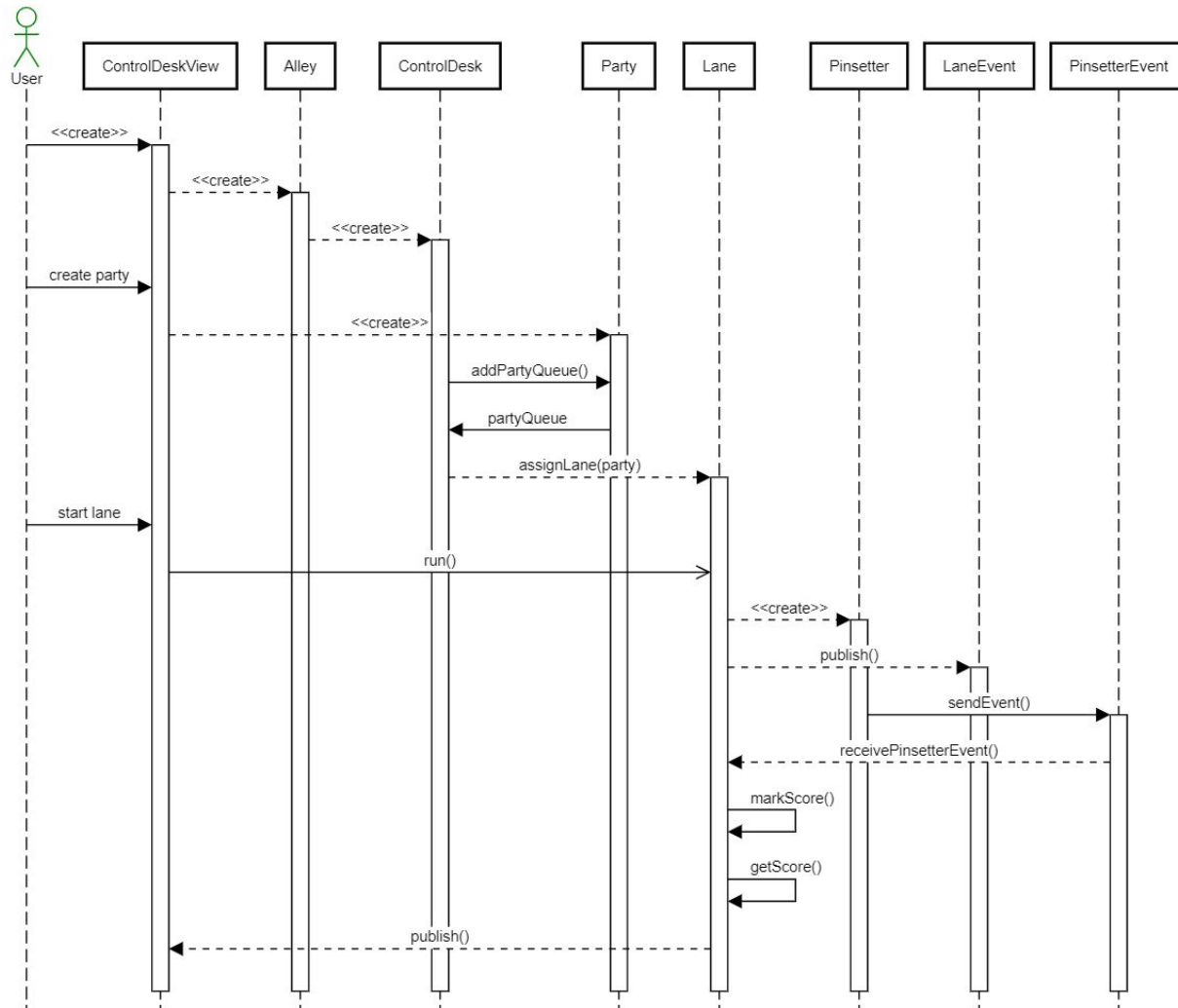


## Before Refactoring

Lane.java “god class” internal behavior and logic

Simple in terms of class communication upon execution

# Running a Lane Before Refactoring





# Lane.run() Post Refactoring

State pattern implemented

Offloaded logic

Increased cohesion





User

ControlDeskView

Alley

ControlDesk

Party

Lane

Game

PlayingGameState

Pinsetter

PinsetterEvent

start lane

run()

<<create>>

<<create>>

<<create>>

begin()

nextTurn()

nextTurn()

ballThrown()

sendEvent()

receivePinsetterEvent()

getCurrentThrower()

currentThrower

markScore()

getScore()

publish()



## Lane.getScore() Post Refactoring

State pattern implemented

Much offloaded logic

Lane calls another class which handles everything in concrete states



User



start lane

run()

getScore()

<<create>>

calculateTotal()

<<create>>

alt

[strike]

<<create>>

calculateTotal()

totalScore

[double strike]

<<create>>

calculateTotal()

totalScore

[spare]

<<create>>

calculateTotal()

totalScore

[normal]

<<create>>











calculateTotal()

totalScore

score

publish()

## Conclusion and Final Metrics

method	CogC	ev(G)	iv(G) ▼	v(G)
  PartyIterator.process(LaneEvent, JLabel[], JLabel[][])	21	1	11	13
  Lane.receivePinsetterEvent(PinsetterEvent)	25	1	9	12
  AddPartyView.actionPerformed(ActionEvent)	16	1	10	11
  LaneStatusView.actionPerformed(ActionEvent)	17	1	9	9
  Lane.formatScores(int[])	20	1	6	8

Top five complex metrics show vast improvements in our code structure. We were able to tackle four out of the five metrics we set out to change from our original plan.