Software Engineering Assignment - 2

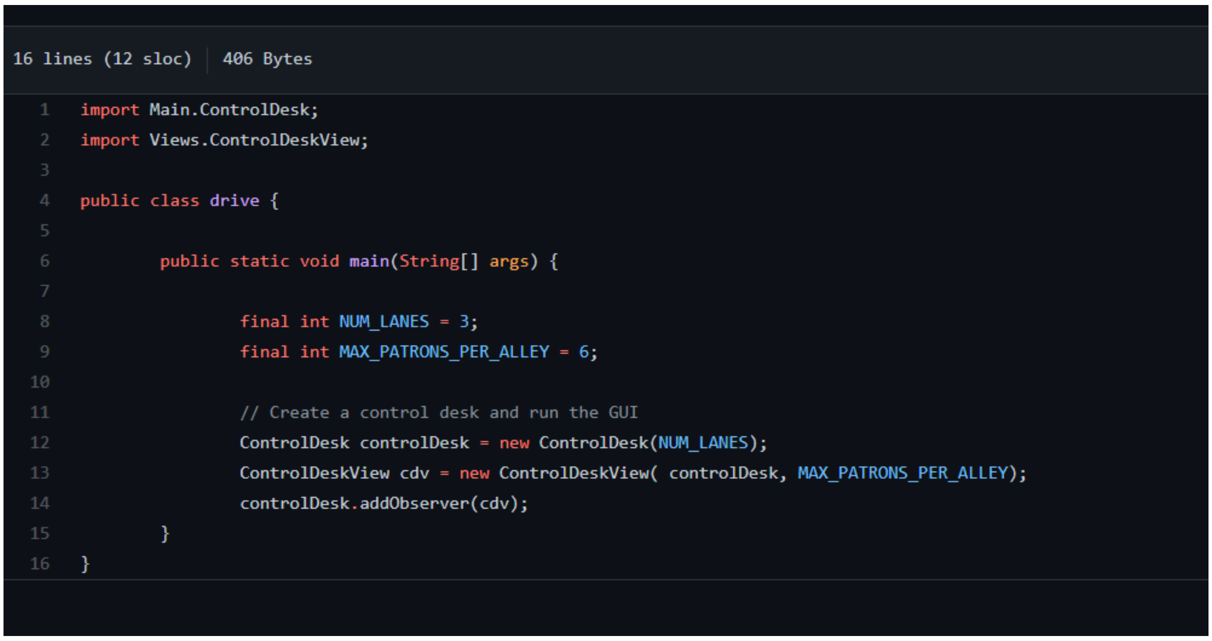
Adding New Features to Bowling Alley Simulation

Team - 15

Overview of the New Features Implemented:

Made the Code extensible for a maximum of 6 players

We made the code adaptable for a maximum of 6 players. At first only 5 players were allowed at maximum. We changed a bit of code and made it compatible for the maximum of 6 players. Following is the snapshot of the code



16 lines (12 sloc) | 406 Bytes

```java
import Main.ControlDesk;
import Views.ControlDeskView;

public class drive {

        public static void main(String[] args) {

                final int NUM_LANES = 3;
                final int MAX_PATRONS_PER_ALLEY = 6;

                // Create a control desk and run the GUI
                ControlDesk controlDesk = new ControlDesk(NUM_LANES);
                ControlDeskView cdv = new ControlDeskView( controlDesk, MAX_PATRONS_PER_ALLEY);
                controlDesk.addObserver(cdv);
        }
}
```

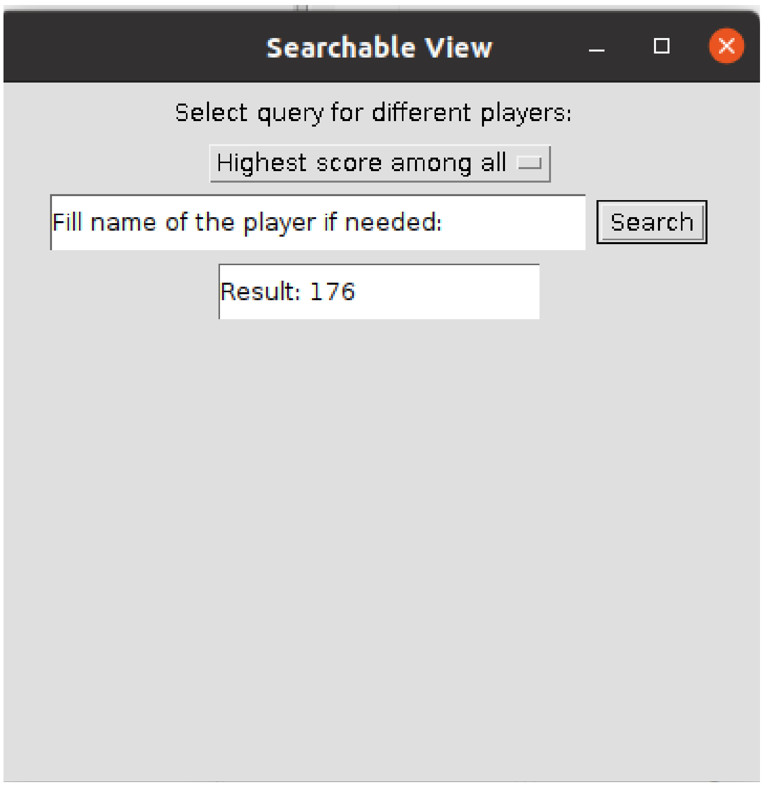
Implemented a database layer which supports various queries

We have implemented the database layer by creating a new class called searchDATABASE.
We created a new view file named SearchView.java in which we have implemented the database class
The following features can be accessed from the database class of ours:

- Maximum score
- Minimum score
- Specific score of a particular candidate
- Last 5 score
- Highest and lowest score of a specific candidate.

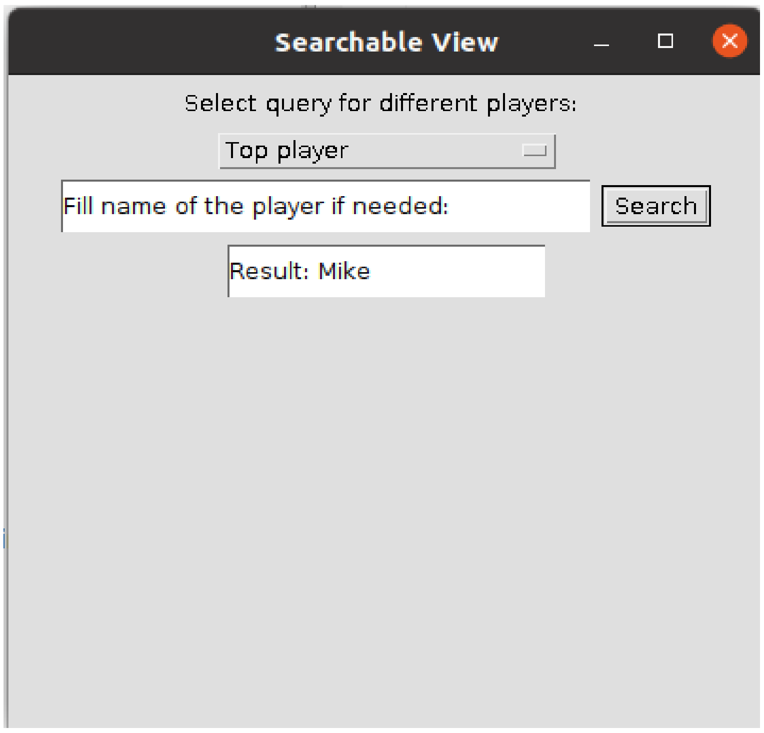
We have also implemented the corner case that if by chance any specific candidate is not there in the playing arena then no data will be shown of them and an error message will be shown.
Below are the few snapshots of the database part implemented.

Displaying highest score among all:


Searchable View
Select query for different players:
Highest score among all
Fill name of the player if needed:
Search

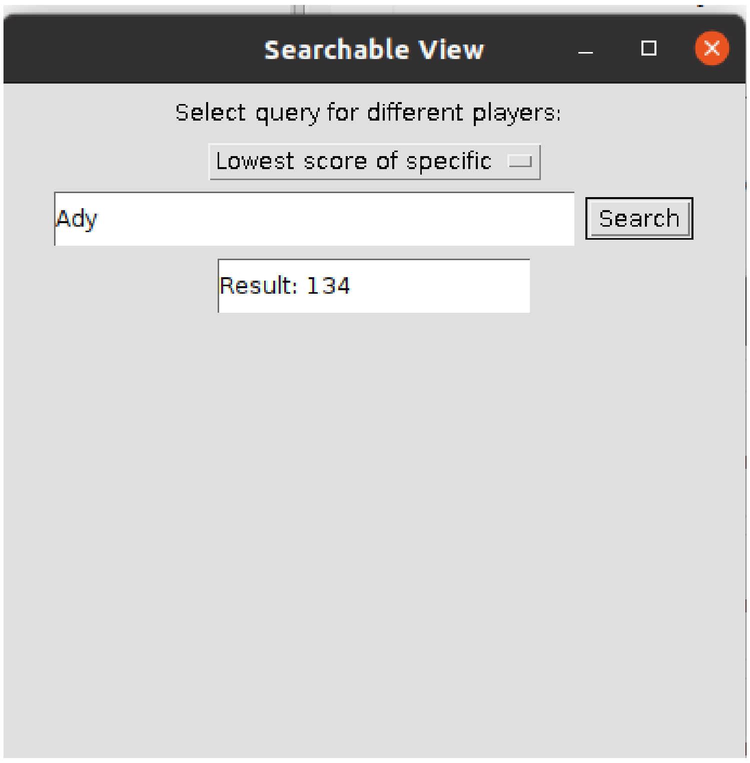
Result: 176

Displaying the top player:


Searchable View
Select query for different players:
Top player
Fill name of the player if needed:
Search

Result: Mike

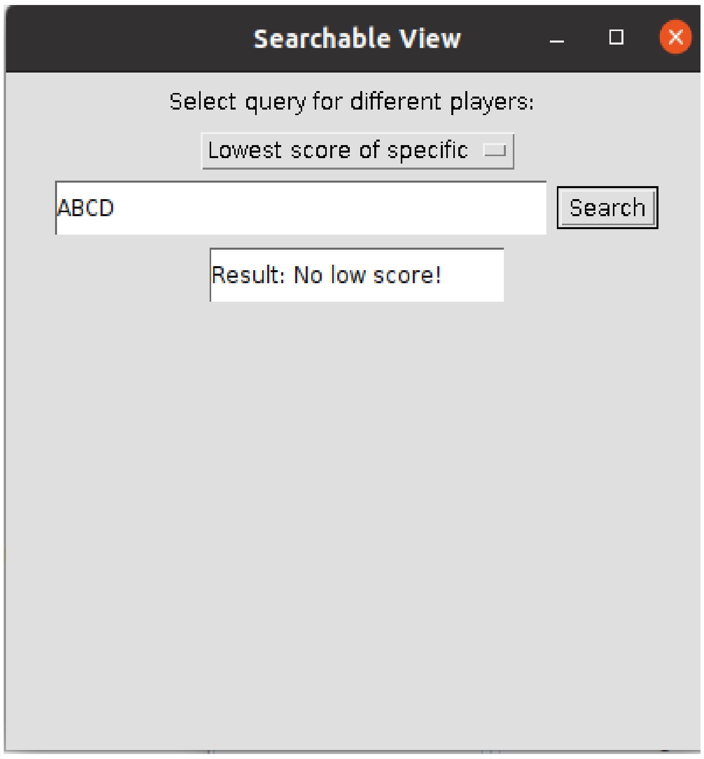
Displaying player specific query:



Ady

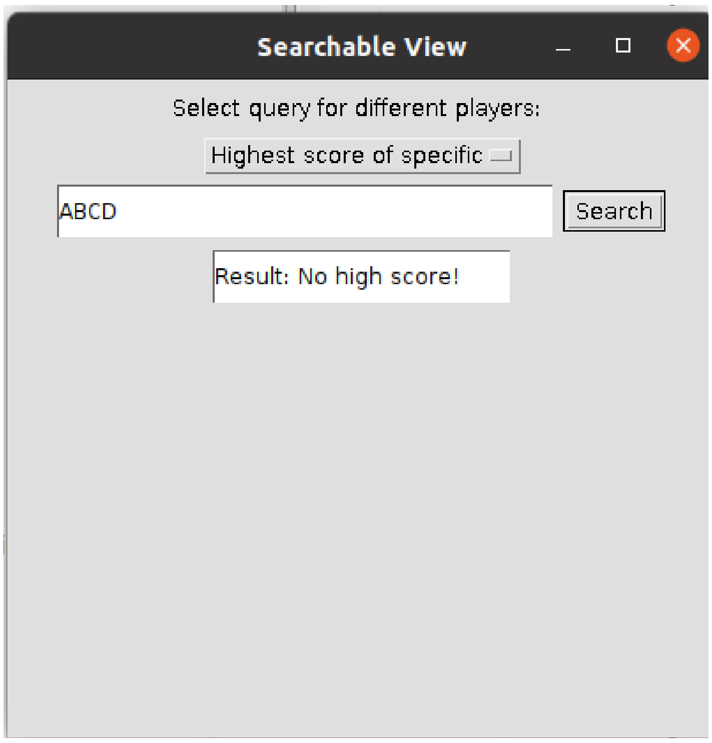
Result: 134

Error handling on wrong queries:



ABCD

Result: No low score!


Searchable View
Select query for different players:
Highest score of specific
Search

ABCD

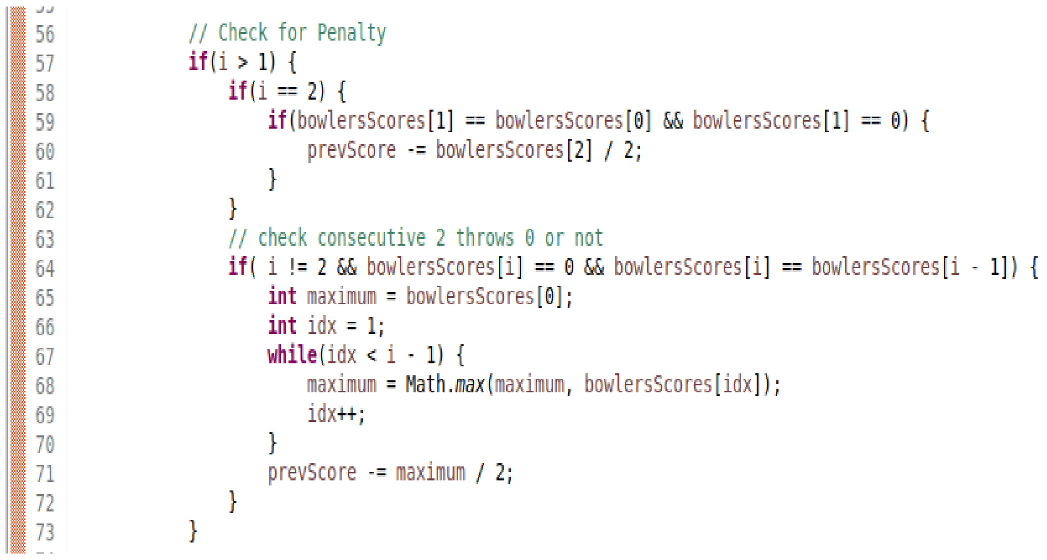
Result: No high score!

Implemented a new penalty system for Gutters

We have implemented the change penalty system. The penalty system is as follows:

- Assuming that the nth and (n-1)th throw resulted in the gutter, then the penalty will be to see the maximum score from 0 to (n-2)th throw and then deduct the marks from that score which will be equivalent to the half of that score
- Assuming that the 0th and 1th throw results in the gutter, then the penalty is calculated by deducting the score from the 2nd throw which will be equivalent to the half of the score scored in 2nd throw.

The code snippet for its implementation is given below:



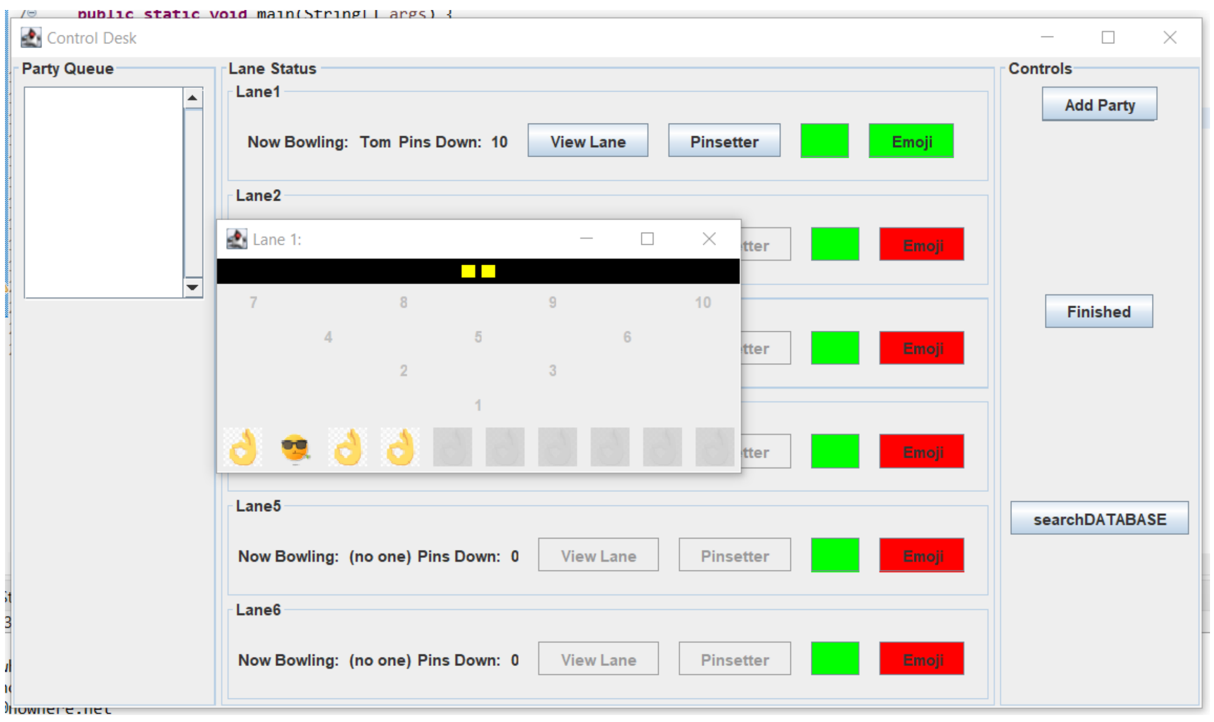
```
56          // Check for Penalty
57          if(i > 1) {
58              if(i == 2) {
59                  if(bowlersScores[1] == bowlersScores[0] && bowlersScores[1] == 0) {
60                      prevScore -= bowlersScores[2] / 2;
61                  }
62              }
63              // check consecutive 2 throws 0 or not
64              if( i != 2 && bowlersScores[i] == 0 && bowlersScores[i] == bowlersScores[i - 1]) {
65                  int maximum = bowlersScores[0];
66                  int idx = 1;
67                  while(idx < i - 1) {
68                      maximum = Math.max(maximum, bowlersScores[idx]);
69                      idx++;
70                  }
71                  prevScore -= maximum / 2;
72              }
73          }
```

(Lines 56-73 is the part where the penalty code is executed)

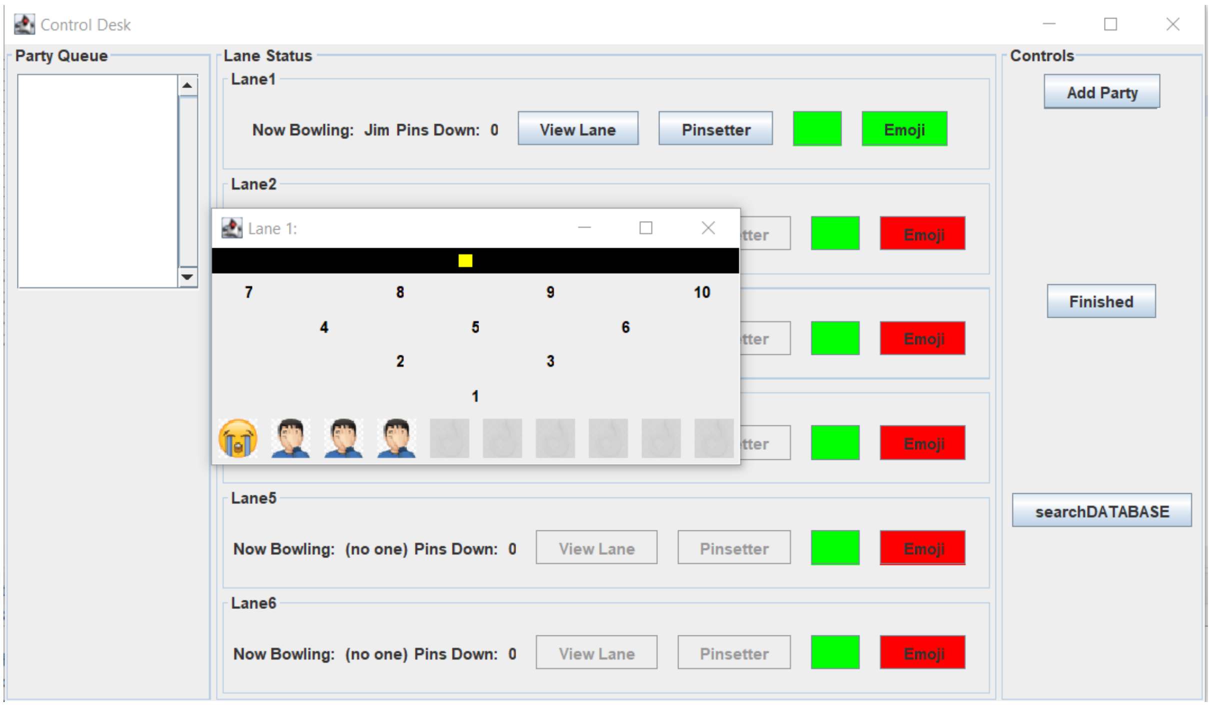
Implemented a emoticon system which provides feedback to the user

We implemented a new system of emoticons which serves as a feedback to the user. Many different types of emojis have been used which are shown depending on the number of pins that fall during a throw.

Snapshots of the emoticon system have been shown below:



Situation when player makes a strike.

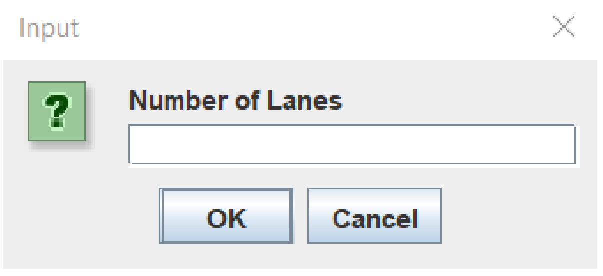


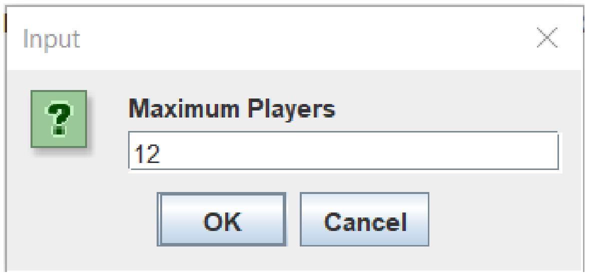
Situation when player misses.

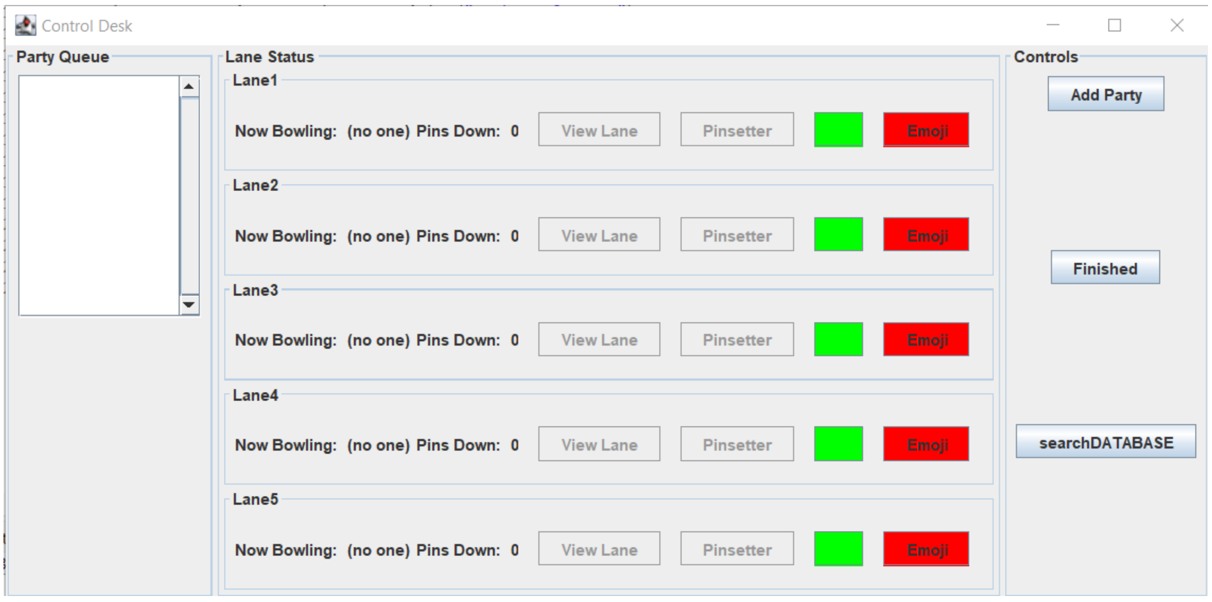
We have implemented this by modifying the PinSetterView and LaneView classes. We see by the implementation of such a system the user is provided with a constant feedback each time he bowls.

Made the code as configurable and Interactive as possible

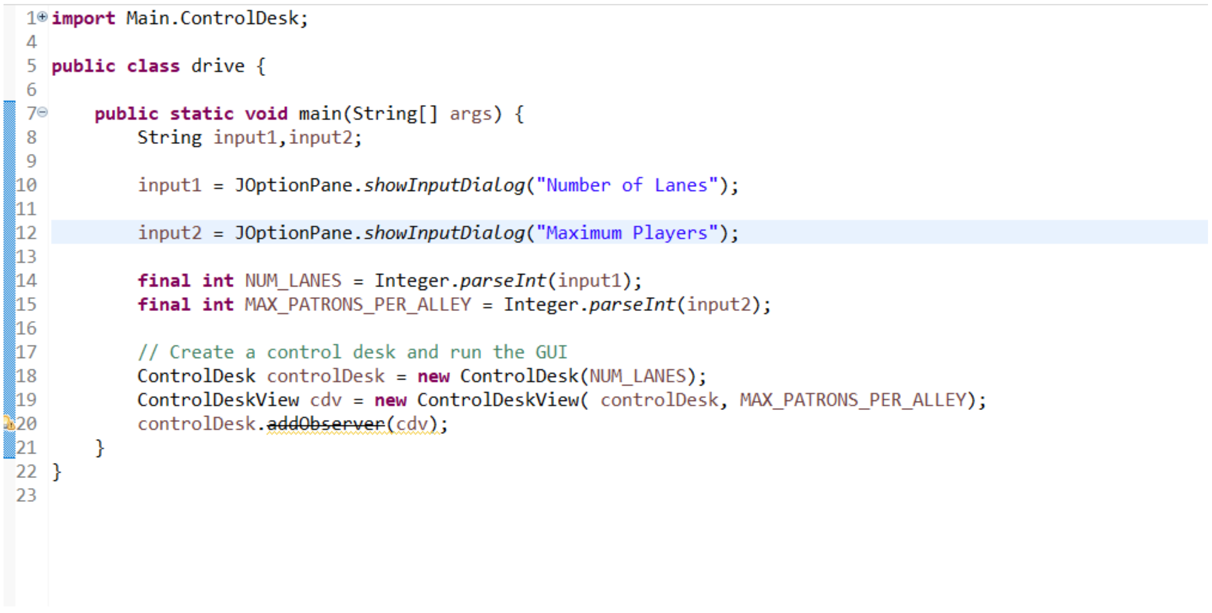
We also tried to make the code as configurable as possible by taking the **Number of Lanes** and **Maximum Players** in each party as a user input.







The code snippet of this change has been shown below:



```
import Main.ControlDesk;

public class drive {

    public static void main(String[] args) {
        String input1,input2;

        input1 = JOptionPane.showInputDialog("Number of Lanes");

        input2 = JOptionPane.showInputDialog("Maximum Players");

        final int NUM_LANES = Integer.parseInt(input1);
        final int MAX_PATRONS_PER_ALLEY = Integer.parseInt(input2);

        // Create a control desk and run the GUI
        ControlDesk controlDesk = new ControlDesk(NUM_LANES);
        ControlDeskView cdv = new ControlDeskView( controlDesk, MAX_PATRONS_PER_ALLEY);
        controlDesk.addObserver(cdv);
    }
}
```

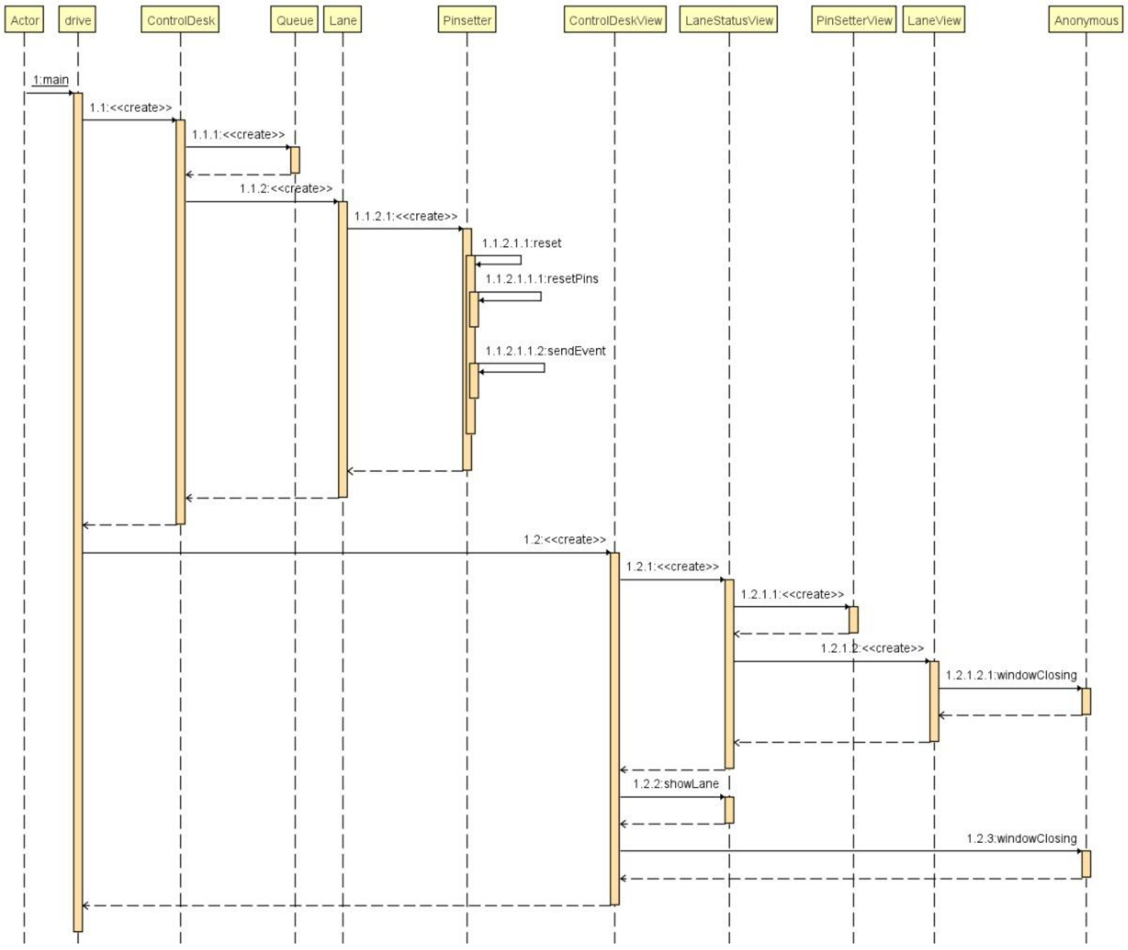
User Design Patterns used

1. User Interface Design Pattern - We have implemented an Emoticon system which gives feedback to each bowler. Also, we use drop-downs and menus to show the result of searchDatabase. These are forms of User Interface Design Pattern.

UML Class and Sequence Diagram for the updated code

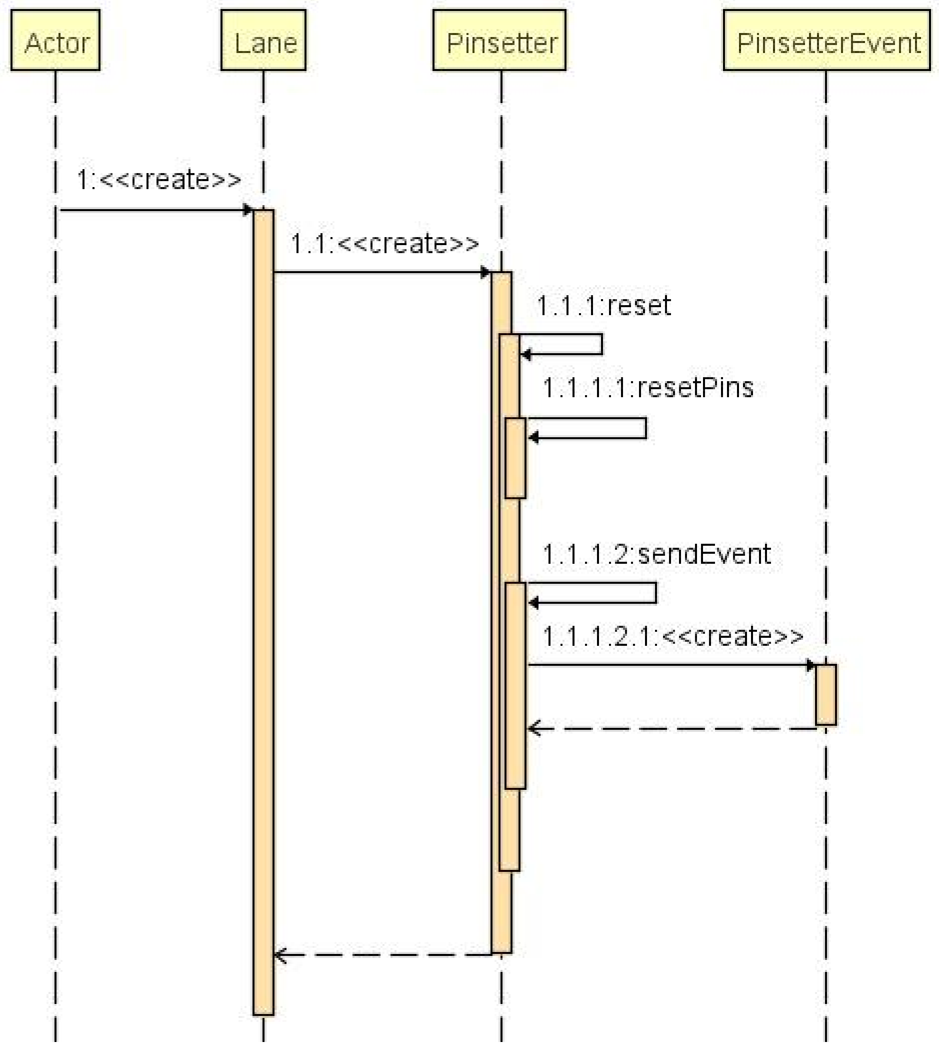
Sequence Diagram of the Updated Code

Drive.java

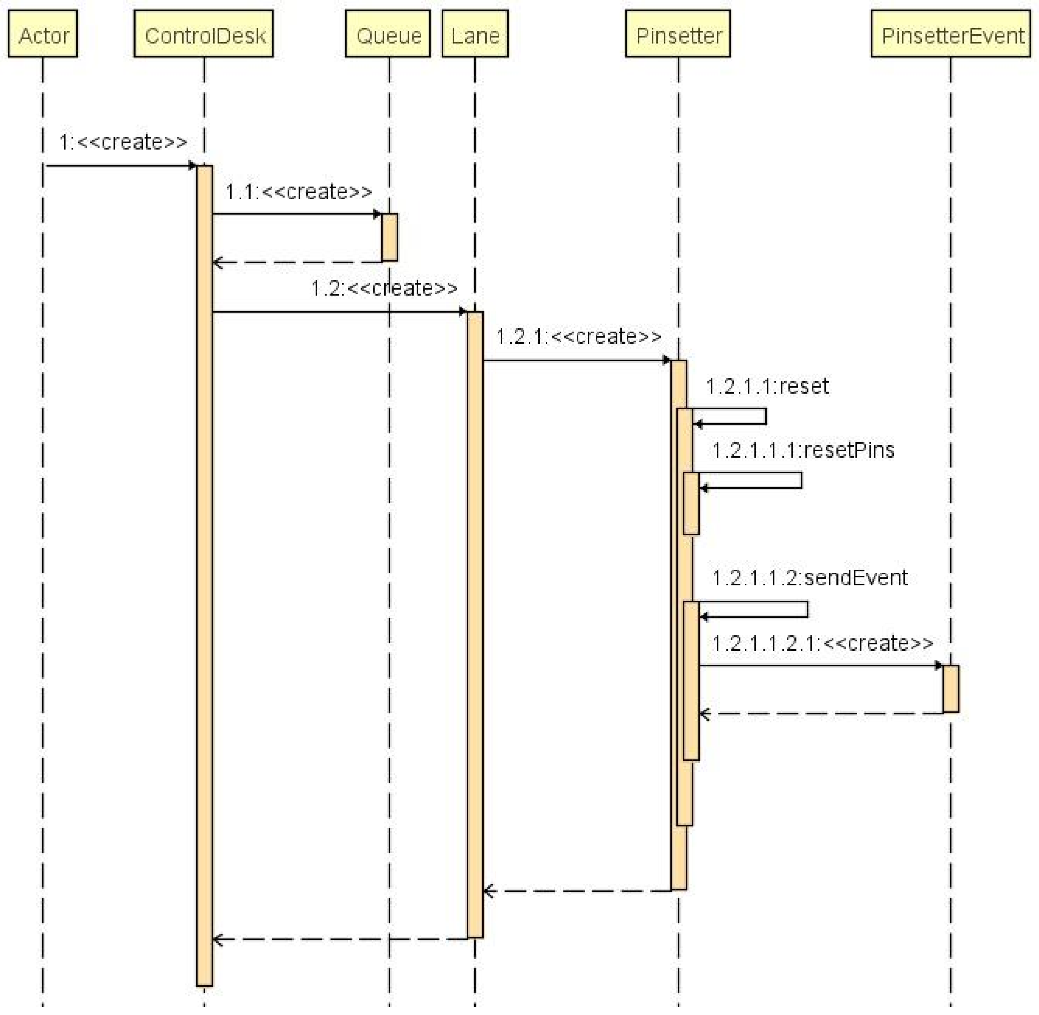


Actor
drive
ControlDesk
Queue
Lane
Pinsetter
ControlDeskView
LaneStatusView
PinSetterView
LaneView
Anonymous
1:main
1.1:<<create>>
1.1.1:<<create>>
1.1.2:<<create>>
1.1.2.1:<<create>>
1.1.2.1.1:reset
1.1.2.1.1.1:resetPins
1.1.2.1.1.2:sendEvent
1.2:<<create>>
1.2.1:<<create>>
1.2.1.1:<<create>>
1.2.1.2:<<create>>
1.2.1.2.1:windowClosing
1.2.2:showLane
1.2.3:windowClosing

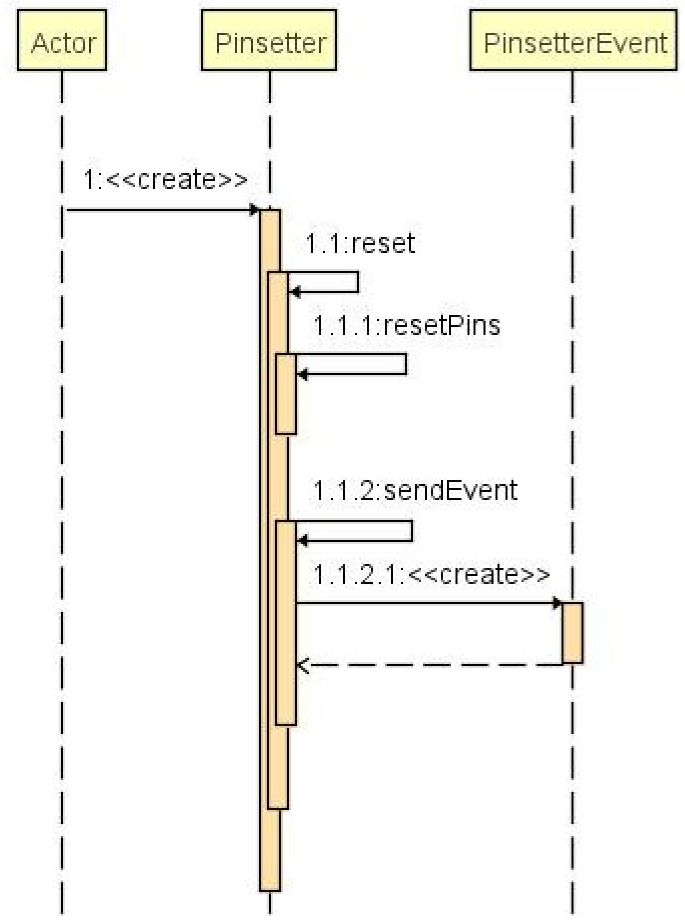
Lane.java


Actor
Lane
Pinsetter
PinsetterEvent
1:<<create>>
1.1:<<create>>
1.1.1:reset
1.1.1.1:resetPins
1.1.1.2:sendEvent
1.1.1.2.1:<<create>>

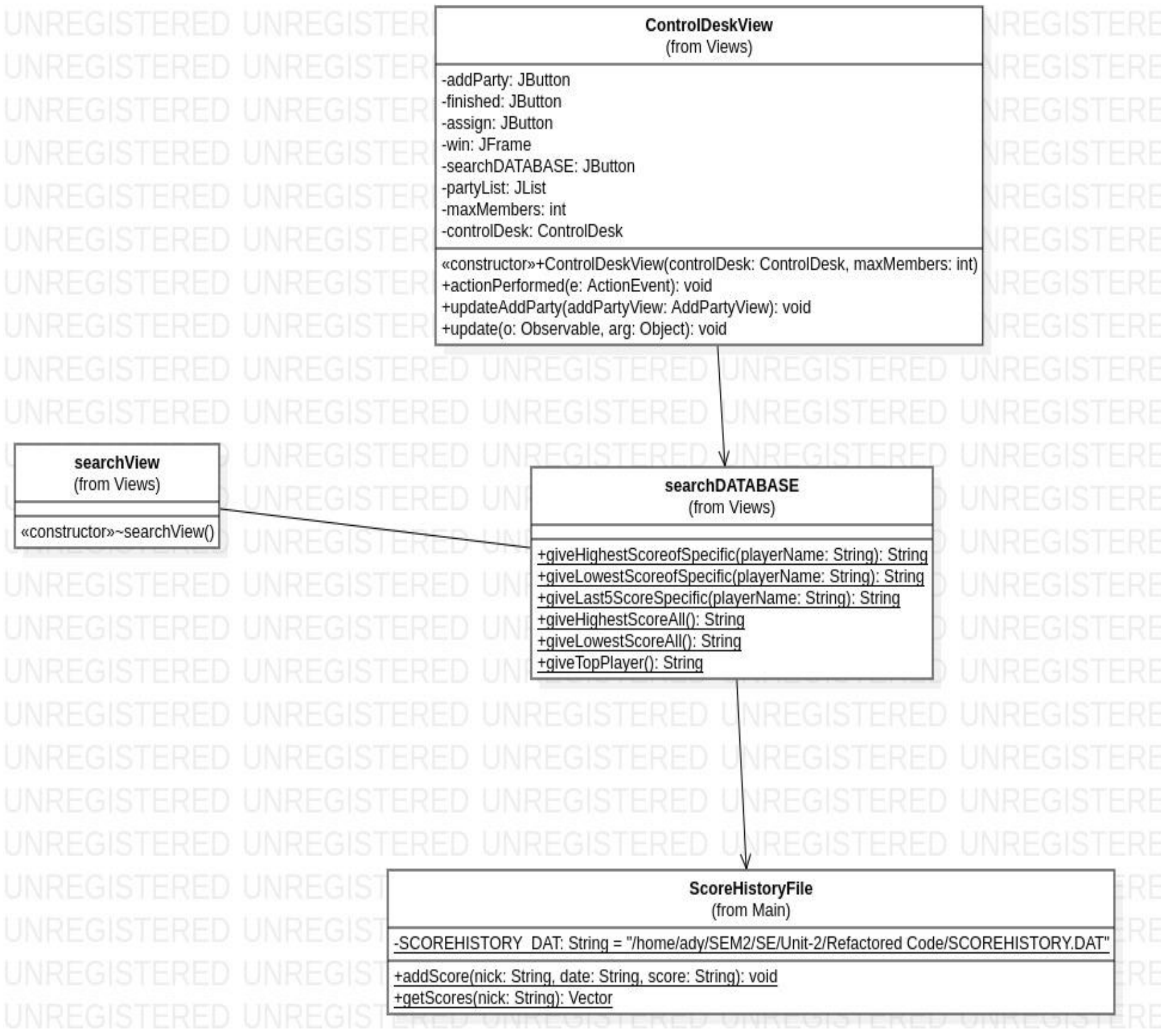
ControlDesk.java



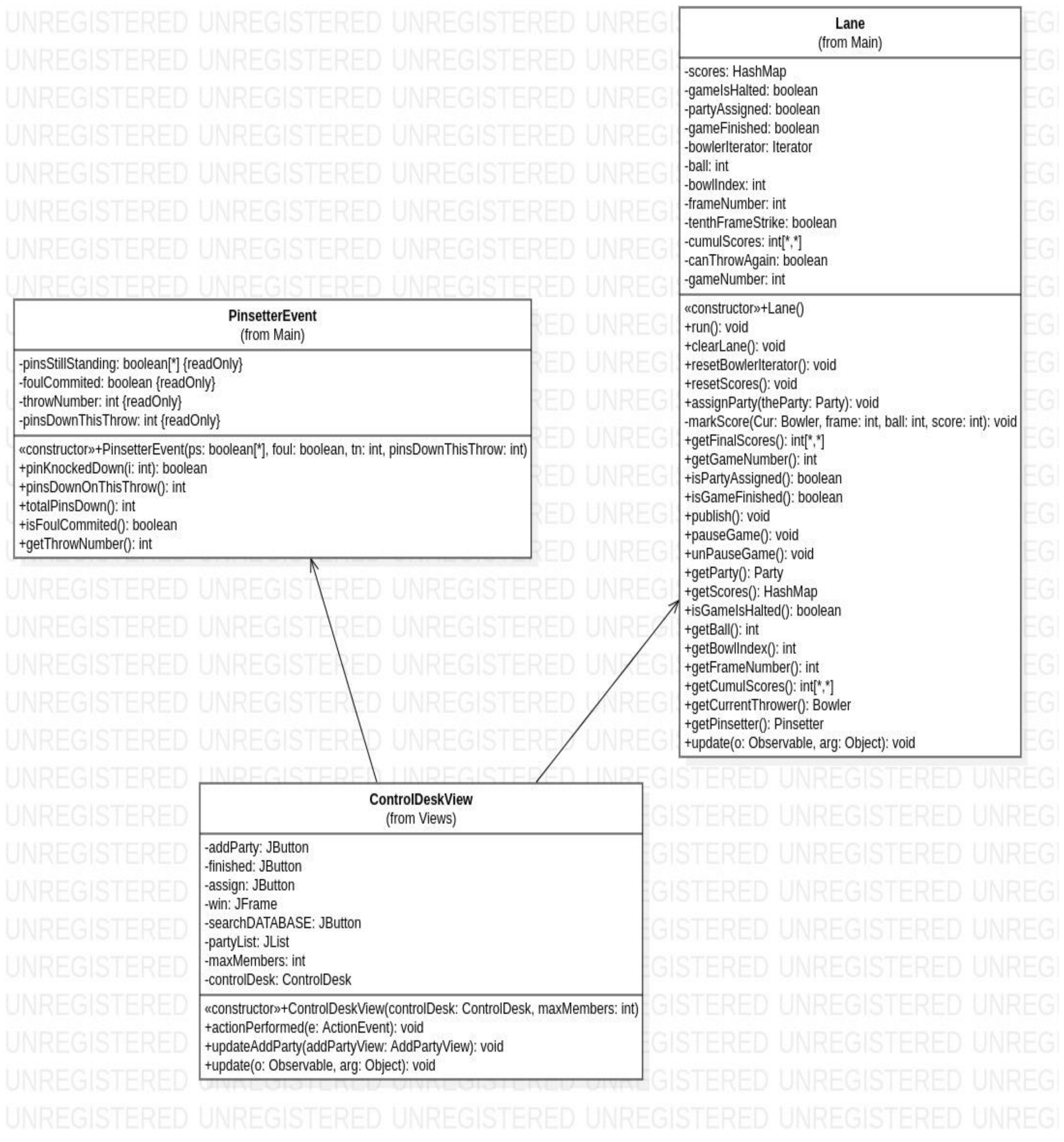
PinSetter.java



Class diagram for database integration


ControlDeskView
(from Views)
-addParty: JButton
-finished: JButton
-assign: JButton
-win: JFrame
-searchDATABASE: JButton
-partyList: JList
-maxMembers: int
-controlDesk: ControlDesk
«constructor»+ControlDeskView(controlDesk: ControlDesk, maxMembers: int)
+actionPerformed(e: ActionEvent): void
+updateAddParty(addPartyView: AddPartyView): void
+update(o: Observable, arg: Object): void
searchView
(from Views)
«constructor»~searchView()
searchDATABASE
(from Views)
+giveHighestScoreofSpecific(playerName: String): String
+giveLowestScoreofSpecific(playerName: String): String
+giveLast5ScoreSpecific(playerName: String): String
+giveHighestScoreAll(): String
+giveLowestScoreAll(): String
+giveTopPlayer(): String
ScoreHistoryFile
(from Main)
-SCOREHISTORY_DAT: String = "/home/ady/SEM2/SE/Unit-2/Refactored Code/SCOREHISTORY.DAT"
+addScore(nick: String, date: String, score: String): void
+getScores(nick: String): Vector

Refined class diagram for emoji feedback:


Lane
(from Main)
-scores: HashMap
-gameIsHalted: boolean
-partyAssigned: boolean
-gameFinished: boolean
-bowlerIterator: Iterator
-ball: int
-bowlIndex: int
-frameNumber: int
-tenthFrameStrike: boolean
-cumulScores: int[*,*]
-canThrowAgain: boolean
-gameNumber: int
«constructor»+Lane()
+run(): void
+clearLane(): void
+resetBowlerIterator(): void
+resetScores(): void
+assignParty(theParty: Party): void
-markScore(Cur: Bowler, frame: int, ball: int, score: int): void
+getFinalScores(): int[*,*]
+getGameNumber(): int
+isPartyAssigned(): boolean
+isGameFinished(): boolean
+publish(): void
+pauseGame(): void
+unPauseGame(): void
+getParty(): Party
+getScores(): HashMap
+isGameIsHalted(): boolean
+getBall(): int
+getBowlIndex(): int
+getFrameNumber(): int
+getCumulScores(): int[*,*]
+getCurrentThrower(): Bowler
+getPinsetter(): Pinsetter
+update(o: Observable, arg: Object): void
PinsetterEvent
(from Main)
-pinsStillStanding: boolean[*] {readOnly}
-foulCommited: boolean {readOnly}
-throwNumber: int {readOnly}
-pinsDownThisThrow: int {readOnly}
«constructor»+PinsetterEvent(ps: boolean[*], foul: boolean, tn: int, pinsDownThisThrow: int)
+pinKnockedDown(i: int): boolean
+pinsDownOnThisThrow(): int
+totalPinsDown(): int
+isFoulCommited(): boolean
+getThrowNumber(): int
ControlDeskView
(from Views)
-addParty: JButton
-finished: JButton
-assign: JButton
-win: JFrame
-searchDATABASE: JButton
-partyList: JList
-maxMembers: int
-controlDesk: ControlDesk
«constructor»+ControlDeskView(controlDesk: ControlDesk, maxMembers: int)
+actionPerformed(e: ActionEvent): void
+updateAddParty(addPartyView: AddPartyView): void
+update(o: Observable, arg: Object): void