New Features Report

Kanish Anand, Mehul Goyal, Shanmukh Karra

Descriptions of the new classes and functions

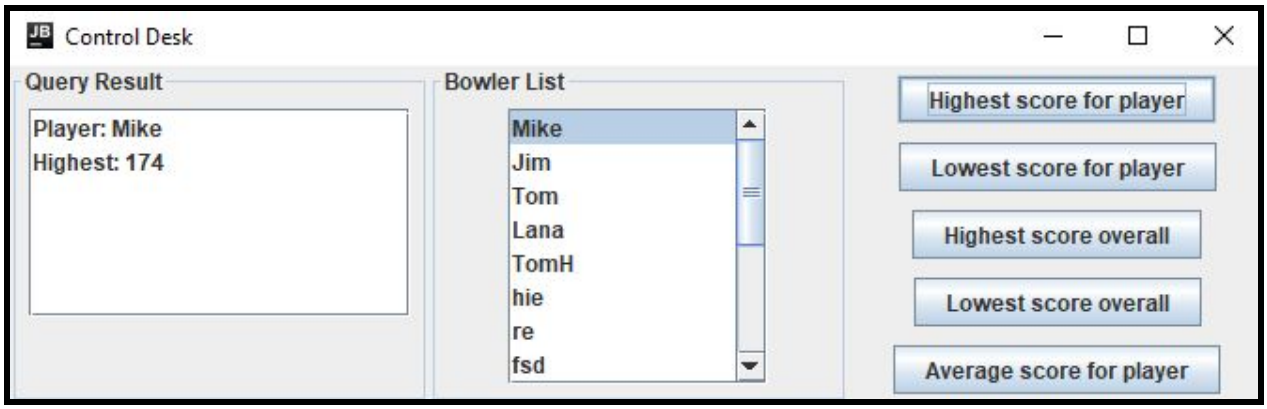
Feature 1 (multiplayer)

```
public class drive {

  public static void main(String[] args) {
      int numLanes = 3;
      int maxPatronsPerParty=6;
  }
}
```

maxPatronsPerParty was changed to 6, which allowed the game to have 6 players, as required.
The option to add and store players' names was already implemented in the codebase, and was tested to ensure that newPatron was working as required.

Feature 2 (database layers + queries)

A new class was created for this feature (QueryView), which shows the view that allows the user to generate queries.



This class allows the user to select the bowler, and view his results. It also allows for overall results to be displayed.

This class implements five different buttons for five queries:

- Highest score for player (ad-hoc)
- Lowest score for player (ad-hoc)
- Highest score overall
- Lowest score overall
- Average score for player (ad-hoc)

Each of these buttons relies on the actionPerformed() function to call the appropriate function in the ScoreHistoryFile class to parse the DAT file and retrieve the queries.

ScoreHistoryFile has the following functions:

- `public static Vector getHighestAndLowest(String nick, boolean isGeneral)`
- `public static double averageScore(String nick)`

These functions refer to the `SCOREHISTORY.DAT` file, read it using BufferedReader, parse it using regex, and retrieve the appropriate values.

```
BufferedReader in = new BufferedReader(new FileReader(SCOREHISTORY_DAT));
String data;
while ((data = in.readLine()) != null) {
        String[] scoredata = data.split("\t");
        scores.add(new Score(scoredata[0], scoredata[1], scoredata[2]));
}
```

The scores are then processed and iterated through, and based on the query, the appropriate result is calculated (max, min, average, highest, lowest, etc).

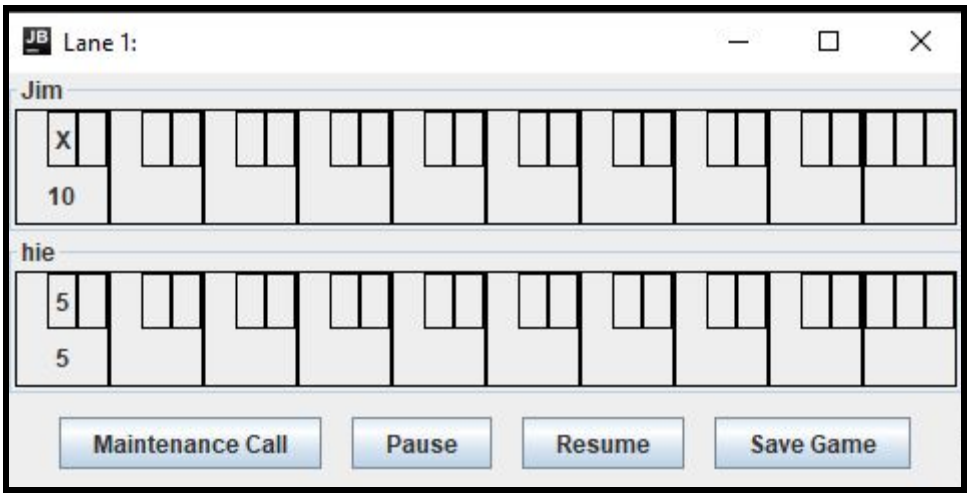
```
Vector toreturn = new Vector<>();
toreturn.add(max);
toreturn.add(bestplayer);
toreturn.add(min);
toreturn.add(worstplayer);
```

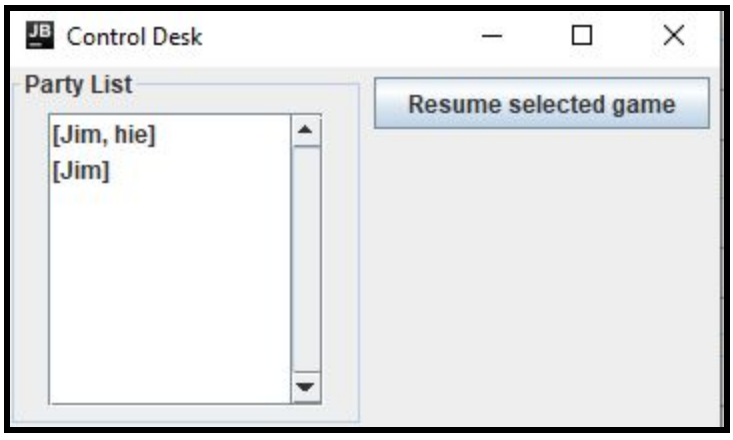
A vector with the required details is returned, and processed in QueryView.

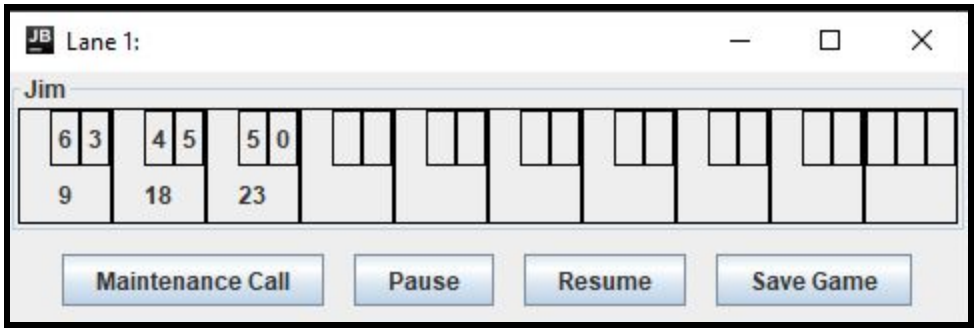
Feature 3 (Pause + resume, persistent saving)

For this feature, two new classes were created:

- ResumeView
    - Creates the window that allows user to select which game to resume
    - Calls functions in PausedLanesFile to retrieve list of all games
    - Calls functions in PausedLanesFile and PartyQueue to resume a game (by removing it from the list of paused games, and adding it to the queue of pending games)
- PausedLanesFile
    - Contains the functions that parse the storage file (LANES.DAT)
    - The addPausedLane() function writes a Lane object in serialized format, using FileOutputStream and ObjectOutputStream








```
public static void addPausedLane(Lane lane) throws IOException,
ClassNotFoundException {
//code
String filepath = "LANE.DAT";
storedLaneVector.add(v);
FileOutputStream fileOut = new FileOutputStream(filepath);
ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
objectOut.writeObject(storedLaneVector);
objectOut.close();

}
```

The addPausedLane function uses serializable objects and writes it into LANE.DAT.

```
public void assignPausedParty( Lane curLane , Vector resumeLane) {
     curLane.setter.rnd = (Random) resumeLane.get(1);
     curLane.setter.pins  = (boolean[]) resumeLane.get(2);
     curLane.setter.foul = (boolean) resumeLane.get(3);
     curLane.setter.throwNumber = (int) resumeLane.get(4);

//     curLane.calculateScore.curScores = new
int[((CalculateScore)((resumeLane).get(5))).party.myBowlers.size()];
//     curLane.calculateScore.finalScores = new
int[((CalculateScore)((resumeLane).get(5))).party.myBowlers.size()][128];
//     curLane.calculateScore.cumulScores = new
int[((CalculateScore)((resumeLane).get(5))).party.myBowlers.size()][10];

     curLane.calculateScore = (CalculateScore) resumeLane.get(5);
     resetBowlerIterator(curLane);

     curLane.gameIsHalted = false;

     curLane.gameFinished = (boolean) resumeLane.get(7);
```

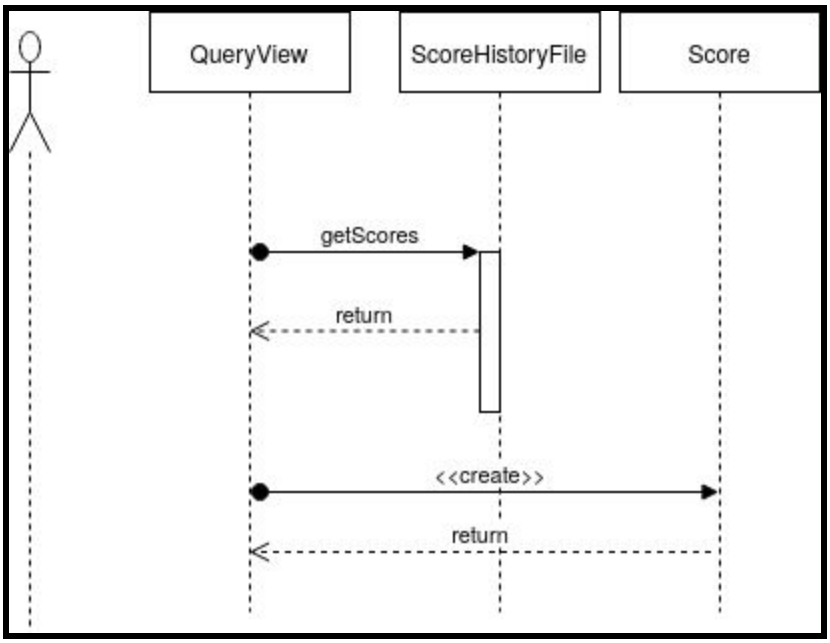
```
     curLane.ball = (int) resumeLane.get(8);
//     curLane.bowlIndex = (int) resumeLane.get(9);
     curLane.bowlIndex = 0;
     curLane.frameNumber = (int) resumeLane.get(10);
     curLane.tenthFrameStrike = (boolean) resumeLane.get(11);
     curLane.canThrowAgain = (boolean) resumeLane.get(12);
     curLane.gameNumber = (int) resumeLane.get(13);
     curLane.currentThrower = (Bowler) resumeLane.get(14);
     System.out.println(curLane.frameNumber + " " + curLane.bowlIndex + " " +
curLane.ball);
  }
```

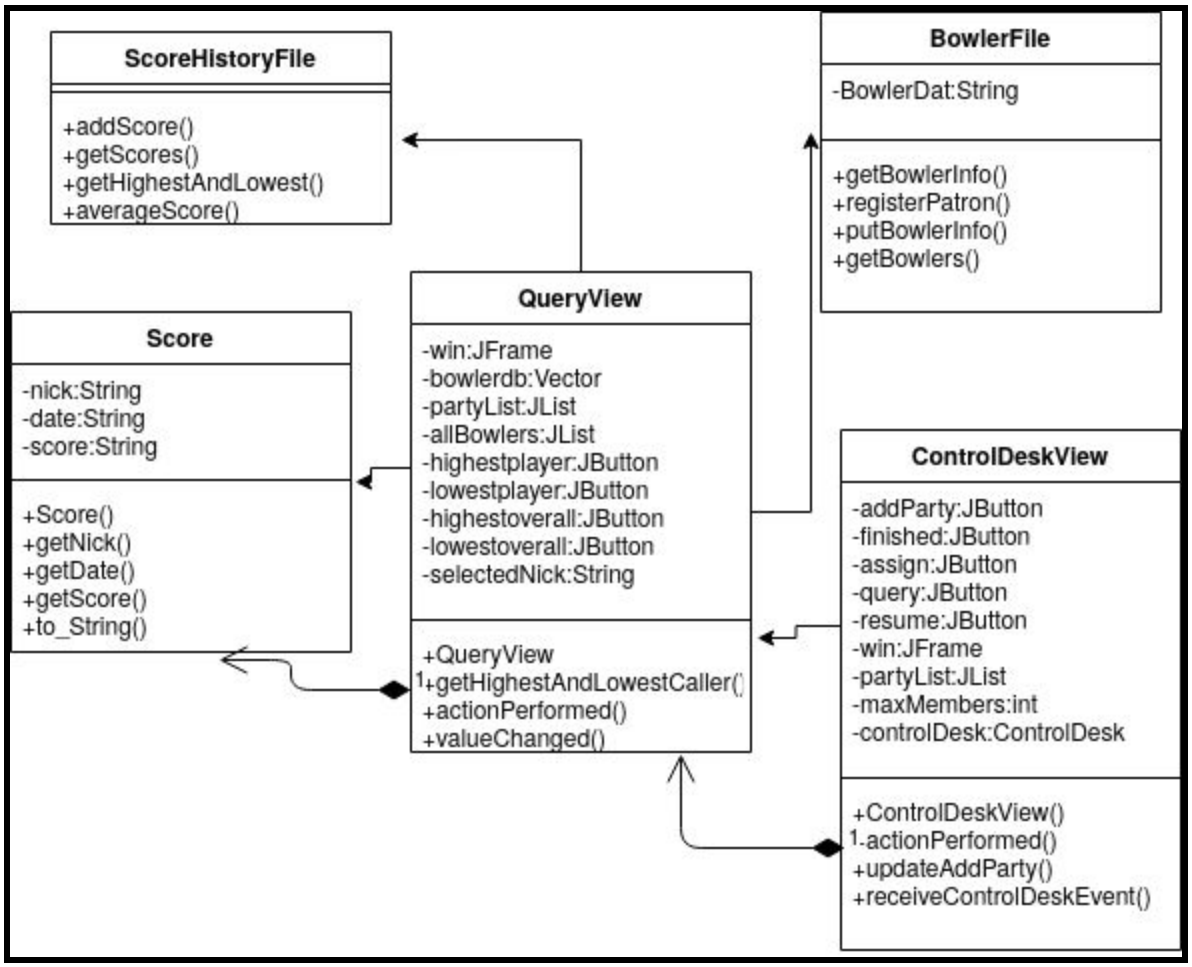
The assignPausedParty function retrieves the data and stores it in curLane's attributes.

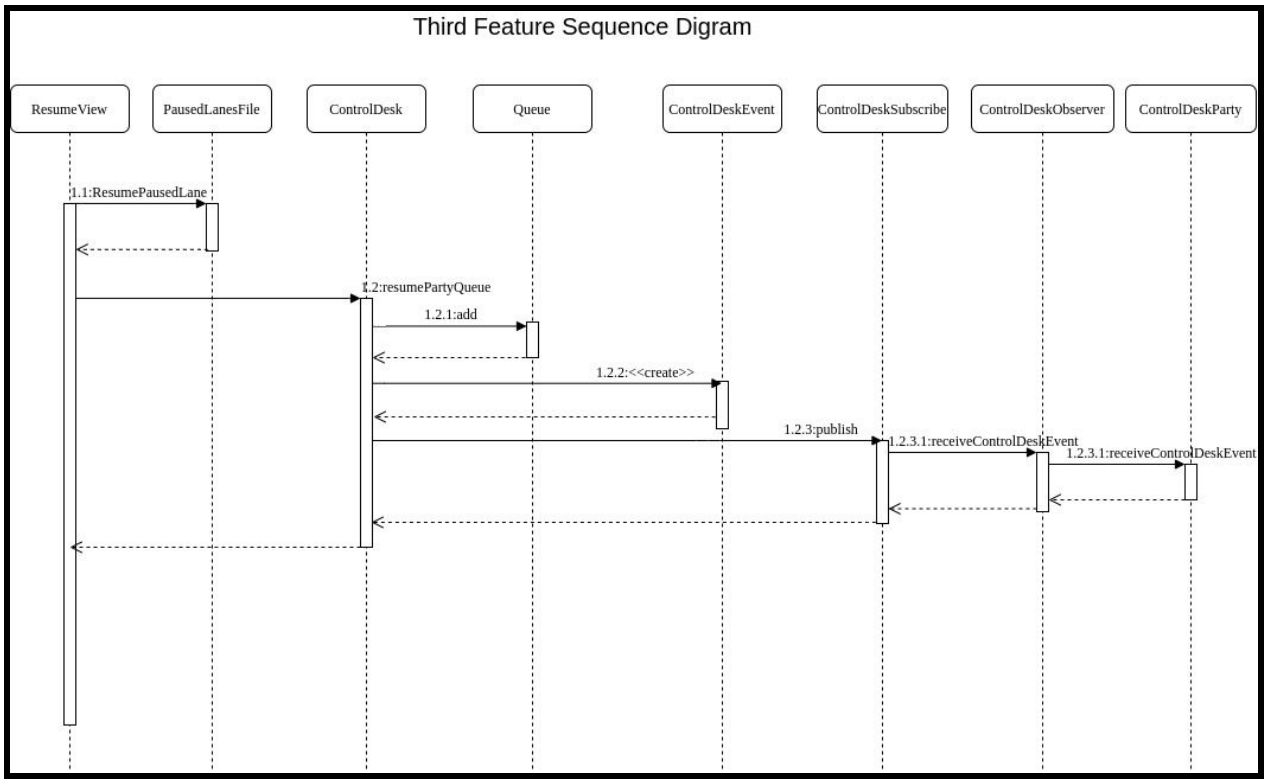
```
while (it.hasNext() && controlDesk.pausePartyQueue.hasMoreElements()) {
   Lane curLane = (Lane) it.next();

   if (!curLane.calculateScore.partyAssigned) {
       System.out.println("ok... assigning this party");
       Vector queueElement = (Vector) controlDesk.pausePartyQueue.next();

       ((Party)queueElement.get(0)).assignPausedParty(curLane, (Vector)
queueElement.get(1));
   }
}
```

The assignLane function has a new while loop which now checks the new pausePartyQueue alongside the existing partyQueue.

UML Class Diagrams and Sequence Diagrams






Third Feature Sequence Digram
ResumeView
PausedLanesFile
ControlDesk
Queue
ControlDeskEvent
ControlDeskSubscribe
ControlDeskObserver
ControlDeskParty
1.1:ResumePausedLane
1.2:resumePartyQueue
1.2.1:add
1.2.2:<<create>>
1.2.3:publish
1.2.3.1:receiveControlDeskEvent
1.2.3.1:receiveControlDeskEvent