Investigating Gender Imbalance in Hollywood Movies

Candice Xie, Joyce Chung, Malahim Tariq

I. Introduction

In 1985, Alison Bechdel introduced the Bechdel Test, which explored the representation of female characters in movies. It has two criteria: the movie has to have at least two named female characters who talk to each other about something other than a man. Surprisingly, a large percentage of films in Hollywood failed this Test. As the white-dominated and male-heavy casts are still the status-quo in Hollywood, our project explores gender inequality and intersectionality in the movie industry.

We created a feminist score, using a combination of the 12 Bechdel-alternative tests, to measure gender imbalance in four categories: people behind the camera, intersectionality, protagonists, and the supporting casts. We chose one or more tests in each category to look into all the people that influenced the creation of the movies, but we also kept in mind that as of now, it is close to impossible for all the movies in Hollywood to pass the Bechdel Test *and* the 12 alternative tests, and to create a reasonable definition for a 'feminist' movie for the society we live in, we chose tests that, in our opinion, had reasonable criteria for a 'feminist' movie.

Through a series of quantitative tests designed by women in the film industry, we combined the Rees-Davies Test, that looks at the gender imbalance behind the camera, the Ko Test, that looks at the intersectionality of the casts, the Peirce, Villarreal, and the Landau Test, that looks into the female protagonists, and the Koeze-Dottle and the Feldman Test, that looks at the gender imbalance of the supporting casts.

II. What is our Criteria for a ‘Feminist’ Movie?

A. Looking Behind the Camera

The Rees-Davies Test

The Rees-Davies Test is one of the 12 alternative Tests to the Bechdel Test, created by **Kate Rees Davies**, a director and producer. This test looks behind the camera by following the criteria of whether each department of the movie has two or more women.

We chose this Test instead of the Uphold or the White Test because we believe that if every department had two or more women, there would be enough encouragement for the women in each department to have a voice with someone else to back them up. Of course, this would not be an accurate measurement to see if there is a fair representation of women behind the scenes, but as of now, we believe that this is a relatively fair criteria to label a movie as a feminist movie or not in combination to the other tests that we chose to define our definition of a ‘feminist’ movie.

We did not choose the Uphold Test because unlike the Rees-Davies Test, it does not check each department, including the behind-the-scenes crew. On the other hand, the White Test has criteria that does check each department like the Rees-Davies Test, but we came to a conclusion that the criteria for this alternative Test was too strict as of now. Although it would be ideal to have half of each department to be women with half of the department heads being women, the Hollywood film industry has yet to reach this point. We wanted to stay true to the reality we live in as of now by creating a definition of a feminist movie that could give moderate “passes” to movies. Also, none of the 50 films have passed the White Test or the Uphold Test, so including these two Test in our feminist score will not have a statistical significance.

B. Intersectionality

The Ko Test

The Ko Test, created by **Naomi Ko**, a writer, and an actress of the movie “Dear White People”, investigates the intersectionality of the cast in Hollywood movies. For a long time, Hollywood movies have been heavily dominated by white actors and actresses. Representation of different ethnicities has increased in Hollywood movies over the years but still lacks to have equal representation. The Ko Test allows movies to pass as a feminist movie if there’s a non-white actress in the movie that has enough screen and speak-time (speaks in 5 or more scenes) and speaks English.

Apart from the Waithe and the Villalobos Test, the Ko Test encompasses all the actresses that are from marginalized racial groups in the film industry, so we decided to only include the Ko Test and not the Waithe and the Villalobos Test. Since representation is very important to how women function in society according to multiracial feminist thoery we gave this test one point.

C. Looking at the Female Protagonists

The Villarreal Test

Lindsey Villarreal, the producer who developed the Villarreal Test, created a solid set of criteria for critiquing female protagonists. The three factors that will cause a movie to fail is if a lead female character is introduced in her first scene as a stereotyped-woman that's sexualized, expressionless and cold, or matriarchal (older, tired, or overworked). However, this test allows the movie to pass if 3 of the following four criteria are met: if the lead female character has a strong career or is in a position of power, is a mother, is reckless, or is identified as someone who is sexual or chooses a sexual identity later in the movie.

We chose this test to represent the female protagonist criteria in our definition of a 'feminist' movie because this test looks at the female protagonist's chracter arc in the story. This test gives the chance for the female protagonist to develop out of the typical, stereotypic women illustrated in the film industry. It is very important that women are seen as more than just objects so we decided to give this test a lot of importance if our feminstscore, so we gave it one point.

Peirce Test

The Peirce Test was created by Kimberly Peirce, the director of "Boys Don't Cry," "Stop-Loss" and "Carrie". This test allows a movie to pass as a feminist movie if there was a female protagonist with her own story, has her own needs and desires that the audience can emphathize with or understand that she pursues.

We weighed this test with the same value as the Villarreal Test because we believe that it is as important for the main female lead to have her own story and voice as much as it is important for her to climb out of a stereotypical woman portrayed in most Hollywood movies if they were portrayed as such. Thus we gave this test one point.

Landau Test

The Landau Test was created by Noga Landau, the writer for "The Magicians". This test fails a movie from passing as a feminist movie if a primary female character ends up dead, pregnant, or causes a plot problem for the male lead.

We wanted to note the importance of not portraying a women in the film with such stereotypes but we also did not give this test the same value as we did for the other two tests in the Protagonists category because we wanted to look more into the story of the female lead and how she develops through the film rather than only one of their actions. A female character could've died or have gotten pregnant in the movie, but that doesn't mean that the film did not allow her to have her own story or break out of the stereotypic character of women in films. We concluded by giving this test partial credit, so that if a movie passes this test, only 0.5 points will be added to the final feminist score rather than 1 full point because we have already included Peirce and Villarreal test which focus in the development of female and main protagonist's development.

D. Looking at the Supporting Casts

Koeze-Dottle Test

The Koeze-Dottle Test was created by Rachael Dottle and Ella Koeze, who are both journalists. The criteria to pass a movie as a feminist movie is that at least fifty percent of the supporting casts of the movie must be women. By "cast" the test refers to supporting actors on the cast list that were paid.

We chose this test because we believe that it's important for women to be part of the cast as "billed supporting actors" instead of as mere cameos or extras. Having women as extras is only the bare minimum for a movie to qualify as a feminist movie. We believe that women should play a significant role in the plot. Passing the Hagen Test doesn't necessarily guarantee this. Being a supporting actor also means you interact in more than one scene, which would give women in film a more influential part to the audience and the story. If the movie passes this test it portrays an accurate representation of the world where more than half of the population are women. Thus we gave this test one point.

Feldman Test

The Feldman Test was created by Rachel Feldman, a director and the former chair of the Directors Guild of America's Women's Steering Committee. This test looks at the overall treatment of women to investigate gender imbalance in films. It qualifies a movie as a feminist movie if it passes with a score of five or higher from the following criteria:
2 points for a female writer or director
1 point for a female composer or director of photography
1 point for three female producers or three female department heads

1 point for a crew that’s 50 percent women
2 points if there’s a female protagonist who determines story outcomes
2 points if no female characters were victimized, stereotyped or sexualized
And 1 point if a sex scene shows foreplay before consummation, or if the female characters initiate or reciprocate sexual advances.

This test evaluates many important criteria which are important in both representation on and behind the camera which our group values so we gave this test one point because it has many overlapping criteria with the other tests that we chose to define feminist movies. By weighing each criteria with different points this test is similar to our own scoring method for the feminist score, and by including this test in our feminist score, we can add another layer of qualification to our feminist movie definition, that allows different movies to build their score towards a passing score. We also included it because its last criteria is very important to counter the misrepresentation of sex by the porn industry.

E. Test Choice Summary

In conclusion, the tests we have chosen to define our definition of a ‘feminist’ movie allow us to get a closer look at both the production side of the movie, its plot, and characters.

The Rees-Davies Test gives us insight into the diversity of various production departments. The Villarreal, Peirce, and Landau Tests allow us to look deeper into the female protagonist in the movies, and the Ko Test accounts for ethnic diversity among the casts. Finally, the Koeze-Dottle Test looks into the gender ratio among the supporting casts of the movies, and the Feldman Test looks at the overall treatment of women.

These are the points we allocated to the tests:

The Rees-Davies Test	1 point
The Villarreal Test	1 point
The Peirce Test	1 point
The Landau Test	0.5 point
The Ko Test	1 point
The Koeze-Dottle Test	1 point
The Feldman Test	1 point

Although these aren't the most strict measures and nowhere near the standards we need to achieve in the industry, we believe that these tests would provide a relatively fair guideline for a feminist movie in our current society.

If Hollywood movies were to follow our definition of a feminist movie, this will help minimize the issues prevalent in Hollywood's film industry, such as the objectification of women by the male gaze and the stereotypical representation of women on-screen. These issues occur when women are not involved at every step of the movie's production, so it's important that Hollywood movies follow a standard to decrease gender imbalance in movies, both on-screen and behind-the-scenes. Thus, as a first step to address gender imbalance in the Hollywood film industry, we have created a definition for a feminist movie.

III. Method

Our program implements the Actor, Movie and MovieCollection classes to explore gender imbalance in movies. The Actor class represents an actor. The Movie class represents a single movie. It implements Comparable<Movie>, which compares the movies based on the feminist score. We defined a method feministScore( ) in this class. Each Movie object will be assigned a feminist score based on the number of alternative Bechdel tests (that we chose to score movies) it passed. We used the data structure, Vector<String>, which contains the result ("1" or "0") for each of the thirteen gender imbalance tests. This information is read from the file "nextBechdel_allTests.csv".

Our feminist score is based on the combination of the Rees-Davies, Villarreal, Peirce, Landau, Koeze-Dottle, Feldman, and the Ko Test. A Movie will have a feminist score of 0 if it did not pass any of these tests and have a score up to 6.5 depending on how many and which of the tests it passed. As explained before, we analyzed each of the 12 alternative Bechdel tests and gave them a score between 0 and 1, that will then add up to the overall feminist score of a movie. A score of 6.5 means the movie is very feminist. According to the standards of our project, it is intersectional, and women were fairly treated as the protagonists, supporting cast, and behind the camera. A score of 0 means that that movie is one of the least feminist movie in the data set.

The feminist score provides a way to organize the data set so that users could easily sort it by giving the movies with higher feminist scores higher priority. The MovieCollection class represents a collection of movie objects. We wrote a method prioritizeMovies() that returns a PriorityQueue of movies in the provided data based on their feminist score. The PriorityQueue class implements the Queue<T> interface based on a LinkedMaxHeap, which, in turn, implements the MaxHeap interface. Thus the most feminist movie will be the root of the max heap and so on. Ties between objects with the same priority are broken by the first in first out order. Thus, if a movie was read from the file and created first, it will be given a higher priority than a movie with the same score just because it was read first.

If we wanted to include more movies in our data set we would add it to the file we read to create our movie objects. The movie would need to be entered into the file(“nextBechdel_castGender.txt”) with the relevant information in this format "MOVIE","ACTOR","CHARACTER_NAME","TYPE","BILLING","GENDER" to our CastGenderFile. We would also need information on if the movie passed each of our chosen tests. We would enter the test results for the movie to the file (“nextBechdel_allTests.txt”) in the order the tests are formatted: “movie name, bechdel, peirce, landau, feldman, villarreal, hagen, ko, villarobos, waithe, koeze_dottle, uphold, white, rees-davies''. Once the data on the movie has been correctly added to these files our movie objects will be created with the following attributes of the movie associated with it – the title of the movie, the actors in the movie (in a hashtable), its tests results (in a vector) and names of the actors (in a linked list), including the order of which it was created (and added to a movie collection; this order is used to break ties between Movies with the same feminist score). The Movie object is a Comparable object thus it can be compared and sorted in different data structures. In our case we use a LinkedList to store all the movies in a MovieCollection object and then later assign a score and sort it into a PriorityQueue. We do not use data structures that have size limits like Arrays so no matter how many Movie objects are added to a MovieCollection object, there won’t be issues of reaching a size limit for the movie collection.

In MovieCollection class, we created methods readMovies(), which reads from the “nextBechdel_allTests.txt” file that contains the information of the movie titles and the corresponding test results, and populates a LinkedList of Movie objects for that collection. The method readCasts() reads from the “nextBechdel_castGender.txt” file that contains information on the cast for each movie, and populates a LinkedList of all the Actors in all the Movies in that collection..

IV. Results

Below are the movies in order of highest to lowest feminist scores from the data set we were given:

```
***Testing prioritizeMovies()***
Movie title: Independence Day: Resurgence, Feminist Score: 5.0, Number of actors: 28
Movie title: Bad Moms, Feminist Score: 4.5, Number of actors: 16
Movie title: Hidden Figures, Feminist Score: 4.5, Number of actors: 96
Movie title: Ghostbusters, Feminist Score: 4.5, Number of actors: 54
Movie title: Finding Dory, Feminist Score: 4.0, Number of actors: 28
Movie title: Allegiant, Feminist Score: 4.0, Number of actors: 0
Movie title: Arrival, Feminist Score: 4.0, Number of actors: 60
Movie title: Sing, Feminist Score: 4.0, Number of actors: 55
Movie title: The Girl on the Train, Feminist Score: 4.0, Number of actors: 20
Movie title: Storks, Feminist Score: 4.0, Number of actors: 25
Movie title: Ice Age: Collision Course, Feminist Score: 3.5, Number of actors: 18
Movie title: Kung Fu Panda 3, Feminist Score: 3.5, Number of actors: 36
Movie title: Miss Peregrine's Home for Peculiar Children, Feminist Score: 3.5, Number of actors: 53
Movie title: The Boss, Feminist Score: 3.5, Number of actors: 37
Movie title: Now You See Me 2, Feminist Score: 3.5, Number of actors: 48
Movie title: Passengers, Feminist Score: 3.5, Number of actors: 27
Movie title: Boo! A Madea Halloween, Feminist Score: 3.0, Number of actors: 42
Movie title: Alice Through the Looking Glass, Feminist Score: 3.0, Number of actors: 23
Movie title: Fantastic Beasts and Where to Find Them, Feminist Score: 3.0, Number of actors: 61
Movie title: La La Land, Feminist Score: 3.0, Number of actors: 49
Movie title: Pete's Dragon, Feminist Score: 3.0, Number of actors: 41
Movie title: Sausage Party, Feminist Score: 3.0, Number of actors: 33
Movie title: Suicide Squad, Feminist Score: 3.0, Number of actors: 54
Movie title: The Conjuring 2: The Enfield Poltergeist, Feminist Score: 3.0, Number of actors: 32
Movie title: The Purge: Election Year, Feminist Score: 3.0, Number of actors: 41
Movie title: Batman v Superman: Dawn of Justice, Feminist Score: 3.0, Number of actors: 122
Movie title: Star Trek Beyond, Feminist Score: 3.0, Number of actors: 23
Movie title: X-Men: Apocalypse, Feminist Score: 2.5, Number of actors: 90
Movie title: Hacksaw Ridge, Feminist Score: 2.5, Number of actors: 90
Movie title: Ride Along 2, Feminist Score: 2.5, Number of actors: 50
Movie title: Sully, Feminist Score: 2.5, Number of actors: 61
Movie title: The Angry Birds Movie, Feminist Score: 2.5, Number of actors: 47
Movie title: The Magnificent Seven, Feminist Score: 2.5, Number of actors: 48
Movie title: 10 Cloverfield Lane, Feminist Score: 2.0, Number of actors: 10
Movie title: Captain America: Civil War, Feminist Score: 2.0, Number of actors: 26
Movie title: Central Intelligence, Feminist Score: 2.0, Number of actors: 25
Movie title: Don't Breathe, Feminist Score: 2.0, Number of actors: 10
Movie title: Lights Out, Feminist Score: 2.0, Number of actors: 14
Movie title: Moana, Feminist Score: 2.0, Number of actors: 7
Movie title: Trolls, Feminist Score: 2.0, Number of actors: 32
Movie title: Zootopia, Feminist Score: 2.0, Number of actors: 32
Movie title: Jason Bourne, Feminist Score: 2.0, Number of actors: 43
Movie title: The Accountant, Feminist Score: 2.0, Number of actors: 58
Movie title: The Legend of Tarzan, Feminist Score: 2.0, Number of actors: 58
Movie title: Teenage Mutant Ninja Turtles: Out of the Shadows, Feminist Score: 1.5, Number of actors: 23
Movie title: The Jungle Book, Feminist Score: 1.5, Number of actors: 11
Movie title: Rogue One: A Star Wars Story, Feminist Score: 1.0, Number of actors: 33
Movie title: Deadpool, Feminist Score: 1.0, Number of actors: 34
Movie title: Doctor Strange, Feminist Score: 1.0, Number of actors: 29
Movie title: The Secret Life of Pets, Feminist Score: 1.0, Number of actors: 31
```

V. Conclusion

According to our results, none of the 50 movies from the dataset that our program read had a feminist score of 6.5. This means that none of the 50 movies passed all 7 of the tests we used to define our feminist score. Although the Hollywood film industry has worked to improve gender imbalance, it has not yet achieved perfect equality, and our results show this. The movie, “Independence Day: Resurgence”, had the highest score of 5.0 while 4 movies (“Rogue One: A Star Wars Story”, “Deadpool”, “Doctor Strange”, and “The Secret Life of Pets”) had the lowest score of 1.0.

We were surprised that movies such as “Hidden Figures” had a feminist score of 4.5, which was high relative to the other movies but not as high as we were expecting it to have (a score of 6.5). This movie is based on three women of African descent that created history as NASA scientists in the 1960s, so we were quite disappointed to find out that this movie did not have a feminist score above 5.0. However, we were able to analyze this movie in more detail, and we were able to find a flaw to our definition of the feminist score.

This movie failed the Koez-Dottle Test, which looks at the gender makeup of the movie’s supporting cast. However, “Hidden Figures” is a movie that depicts the gender imbalance of scientists at NASA during the 1960s, so its supporting casts should be dominated by men. Depending on the story that a movie conveys, it may not necessarily pass some of the criteria for our definition of a feminist movie that heavily based on quantitative data. In the future, we strive to create a test that can look further into the movie’s context.

In addition to the existing 13 Tests, if we were to create a new, alternate Bechdel test, we would create a test that examines the gender pay gap among the lead actors of different genders. This test will compare the average payments received by the leading actors and actresses. According to new research conducted by UWM labor economist John Heywood and his coauthors, **female stars in the nation's movie capital earn an average of $1 million less per film than their male counterparts** when they perform in similar roles (Vickery, 2020).

Another possible, alternative Bechdel test our team would like to propose is one that would analyze the scenes that portray the male gaze towards the female characters in the film. Many times, the camera, that portrays the male character’s gaze, captures the female character’s body in a very sexual way. This objectification of women is much too common in Hollywood films. These shots are very problematic in shaping societal views of women, so if there are scenes like these in the films, this alternative test will fail the movie.

VI. Code

Below is our code for the Actor Class:

```
/**
 * Represents an object of type Actor. An Actor has a name and a gender.
 *
 * @author (Stella K., Joyce, Candice, Malahim)
 * @version (28 Nov 2021, 4:35pm, May 2022)
 */
public class Actor
{
    private String name;
    private String gender;

    /**
     * Constructor for objects of class Actor.
     *
     * @param name of this actor
     * @param gender of this actor
     */
    public Actor(String name, String gender) {
        this.name = name;
        this.gender = gender;
    }

    /**
     * Returns the name of this actor
     *
     * @return the name of this actor
     */
    public String getName() {
        return this.name;
    }

    /**
     * Returns the gender of this actor
     *
     * @return the gender of this actor
     */
    public String getGender() {
        return this.gender;
    }

    /**
     * Sets the name of this actor
     *
     * @param n name of this actor
     */
    public void setName(String n) {
```

```
    this.name = n;
  }

  /**
   * Sets the gender of this actor
   *
   * @param g gender of this actor
   */
  public void setGender(String g) {
    this.gender = g;
  }

  /**
   * This method is defined here because Actor (mutable) is used as a key in a Hashtable.
   * It makes sure that same Actors have always the same hash code.
   * So, the hash code of any object that is used as key in a hash table,
   * has to be produced on an *immutable* quantity,
   * like a String (such a string is the name of the actor in our case)
   *
   * @return an integer, which is the has code for the name of the actor
   */
  public int hashCode() {
    return name.hashCode();
  }

  /**
   * Tests this actor against the input one and determines whether they are equal.
   * Two actors are considered equal if they have the same name and gender.
   *
   * @param other Actor to compare to
   *
   * @return true if both objects are of type Actor,
   * and have the same name and gender, false in any other case.
   */
  public boolean equals(Object other) {
    if (other instanceof Actor) {
      return this.name.equals(((Actor) other).name) &&
      this.gender.equals(((Actor) other).gender); // Need explicit (Actor) cast to use .name
    } else {
      return false;
    }
  }

  /**
```

```
   * Returns a string representation of this Actor.
   *
   * @return a reasonable string representation of this actor, containing their name and gender.
   */
  public String toString() {
    return "Actor " + name + " is a " + gender;
  }

  /**
   * Main method for testing purposes.
   */
  public static void main(String[] args) {
    Actor a1 = new Actor("Charlie", "man");
    Actor a2 = new Actor("Charlie", "man");
    Actor a3 = new Actor("Charles", "woman");

    System.out.println(a1.getName());
    System.out.println(a1.getGender());

    a3.setName("Em");
    a3.setGender("man");
    System.out.println(a3);

    System.out.println(a1.equals(a2));
    System.out.println(a1.equals(a3));
  }
}
```

Below is our code for the Movie Class:

```
import java.io.File;
import java.io.FileNotFoundException;

/**
 * Represents an object of type Movie.
 * A Movie object has a title, some Actors, and results for the twelve Bechdel testResultss.
 *
 * @author (Stella K., Joyce C., Candice X., Malahim T.)
 * @version (28 Nov 2021, 4:35pm; 04/29/2022)
 */
public class Movie implements Comparable<Movie>
{
  private String title;  //title of this movie
private Hashtable<Actor, String> actors;  //collection of actors and their roles
private LinkedList<String> names;  //list of all the actors in this movie
private Vector<String> testResults;  //record of this movie's results for the 12 Bechdel tests

//order on which the Movie was instanted; to break ties in priority queue
private static int nextorder = 0;
private int order;
/**
 * Constructor for objects of class Movie.
 * Initializes instance variables for this movie.
 *
 * @param title of this movie
 */
public Movie(String title) {
  this.title = title;
  actors = new Hashtable<Actor, String>();
  testResults = new Vector<String>();
  names = new LinkedList<String>();
  order = nextorder;
  nextorder++;
}

/**
 * Gets the order of this Movie
 *
 * @return int of the order of this Movie
 */
public int getOrder() {
  return order;
}

/**
 * Gets the title of this Movie
 *
 * @return the title of this Movie
 */
public String getTitle() {
  return this.title;
}

/**
 * Gets this movie's actors (key) along with their
 * roles (values) in a Hashtable
 *
 * @return Hashtable with all the actors who played in this movie
 */
public Hashtable<Actor, String> getAllActors() {
   return actors;
}

/**
 * Gets all the names of the actors in this movie
 * in a Linked List.
 *
 * @return LinkedList with the names of all the actors
 * who played in this movie
 */
public LinkedList<String> getActors() {
   return names;
}

/**
 * Returns a Vector with all the Bechdel test
 * results for this movie. A testResults result can be
 * "1" or "0" indicating whether this move passed
 * or not the corresponding testResults.
 *
 * @return Vector with the Bechdel test results for this movie
 */
public Vector<String> getAllTestResults() {
   return testResults;
}

/**
 * Populates the testResults vector with "0" and "1"s,
 * each representing the result of the coresponding test
 * on this movie. This information will be read from the file
 * "nextBechdel_allTests.csv".
 *
 * @param results  string consisting of of 0's and 1's. Each
 * one of these values denotes the result of the corresponding
 * test on this movie
 */
public void setTestResults(String results) {
   String[] result = results.split(",");
   for(int n = 0; n < result.length; n++) {
      testResults.add(n, result[n]);
      //testResults.set(n, result[n]);  //replace current test results with given test results
```

```
    }
  }

  /**
   * Tests this movie object with the input one and determines whether they are equal.
   *
   * @param other movie to compare to
   *
   * @return true if both objects are movies and have the same title,
   * false in any other case.
   */
  public boolean equals(Object other) {
    if (other instanceof Movie) {
      return this.title.equals(((Movie) other).title); // Need explicit (Movie) cast to use .title
    } else {
      return false;
    }
  }

  /**
   * Takes in a String, formatted as lines are in the input file
   * ("nextBechdel_castGender.txt"), generates an Actor, and adds
   * the object to the actors of this movie.
   *
   * Input String has the following formatting: "MOVIE", "ACTOR","CHARACTER_NAME",
   * "TYPE","BILLING","GENDER"
   *
   * Example of input: "Ricky Dillon","Aspen Heitz","Supporting","18","Male"
   *
   * @param line of information about actor in this movie
   *
   * @return Actor from inputted information
   */
  public Actor addOneActor(String line) {
    //clean the given line to have no quoations on tokens
    line = line.replaceAll("\"", "");

    String[] input = line.split(",");
    String name = input[1];
    String gender = input[5];
    String role = input[2];

    Actor a = new Actor(name, gender);  //create the Actor object with given info from line
    actors.put(a, role);  //add to hashtable of actors with their characters in this movie
    names.add(name);  //add to linked list of actors names in this movie
    return a;
  }

  /**
   * Reads the input file ("nextBechdel_castGender.txt"), and adds all its Actors
   * to this movie.
   *
   * Each line in the movie has the following formatting:
   * Input String has the following formatting: "MOVIE TITLE","ACTOR","CHARACTER_NAME",
   * "TYPE","BILLING","GENDER"
   *
   * Example of input: "Trolls","Ricky Dillon","Aspen Heitz","Supporting","18","Male"
   *
   * @param actorsFile the file containing information on each actor who acted in the movie.
  */
  public void addAllActors(String actorsFile) {
    try {
      //read from file
      Scanner reader = new Scanner(new File(actorsFile));
      //line of info of actor in this movie
      String lineFromFile = "";
      //skip first line that has titles instead of info of actor
      reader.nextLine();
      while (reader.hasNextLine()) { //Continue until reach end of input file
        lineFromFile = reader.nextLine();
        //clean the given line to have no quoations on tokens
        lineFromFile = lineFromFile.replaceAll("\"", "");

        //get title
        String[] infoLine = lineFromFile.split(",");
        String movieTitle = infoLine[0];
        //add the actor to this movie's list if the given line of information is from this movie
        if(movieTitle.equals(this.title)) {
          addOneActor(lineFromFile);
        }
      }
      reader.close(); // Close the file reader
    } catch (FileNotFoundException ex) {
      System.out.println(ex); // Handle FNF by displaying message
    }
  }

  /**
```

```
 * Returns a string representation of this movie for easier testing
 *
 * @return a reasonable string representation of this movie: includes the title,
 * the feminist score, and the number of actors who played in it.
 */
public String toString() {
  String result = "Movie title: " + title
  + ", Feminist Score: " + this.feministScore()
  + ", Number of actors: ";
  if(this.getActors().isEmpty()) {
    result += "0";
  }
  else{
    result += this.getActors().size();
  }
  return result;
}

/**
 * Determines the feminist score of a Movie based on the combination
 * of the Rees-Davies test, Villareal test,the Koeze-Dottle test, the Ko test,
 * the Peirce test, the Landau test, and the Feldman test.
 *
 * The Movie will have a feminist score of ranging between 0 to 6.5 depending
 * on how many of the chosen alternative tests it passed.
 *
 * @return feminist score of this movie based on 7 chosen alternative tests.
 */
public double feministScore() {
  double score = 0.0;
  //indexes of each of the tests we chose to define movies as feminist or not
  int reesdaviesIndex = 12;
  int villarealIndex = 4;
  int koeze_dottleIndex = 9;
  int koIndex = 6;
  int peirceIndex = 1;
  int landauIndex = 2;
  int feldmanIndex = 3;
  if (testResults.get(reesdaviesIndex).equals("0")) {
    score += 1;
  }

  if (testResults.get(villarealIndex).equals("0")) {
    score += 1;
  }

  if (testResults.get(koeze_dottleIndex).equals("0")) {
    score += 1;
  }

  if (testResults.get(koIndex).equals("0")) {
    score += 1;
  }

  if (testResults.get(peirceIndex).equals("0")) {
    score += 1;
  }

  if (testResults.get(landauIndex).equals("0")) {
    score += 0.5;
  }

  if (testResults.get(feldmanIndex).equals("0")) {
    score += 1;
  }
  return score;
}

/**
 * Compares this Movie to the other Movie (parameter) based
 * on their feminist scores that determines their priority.
 * Uses the order in which the elements were added to break ties.
 *
 * @param other Movie to compare to
 *
 * @return int that determines if this Movie has a greater (positive #)
 * or less (negative #) priority than the other Movie
 */
public int compareTo(Movie other) {
  if(this.feministScore() < other.feministScore()) {
    return -1;
  }
  else if(this.feministScore() > other.feministScore()) {
    return 1;
  }
  //if the Movies have the same priority
  //if this Movie was created & added after the other Movie
  else if(this.order > other.getOrder()){
    return -1;
  }
  //if this Movie was created & added before the other Movie
  else {
    return 1;
  }
}

/**
 * Main method for testResultsing.
 */
public static void main(String[] args) {
  Movie m1 = new Movie("Boo! A Madea Halloween");
  Movie m2 = new Movie("Boo! A Madea Halloween");
  Movie m3 = new Movie("Sully");

  System.out.println("TESTING EQUALS() METHOD");
  System.out.println("EXPECTING TRUE: " + m1.equals(m2));  //true
  System.out.println("EXPECTING FALSE: " + m1.equals(m3));  //false

  System.out.println();

  System.out.println("TESTING GETTITLE() METHOD");
  System.out.println(m1.getTitle()); //Boo! A Madea Halloween
  System.out.println(m3.getTitle()); //Sully

  System.out.println();

  System.out.println("ADDING ACTORS FROM MOVIE BOO! A MADEA HALLOWEEN");
  m1.addAllActors("nextBechdel_castGender.txt");
  System.out.println("HASHTABLE OF ACTORS: " + m1.getAllActors());
  System.out.println("LINKEDLIST OF ACTOR NAMES: " + m1.getActors());

  System.out.println();

  System.out.println("SETTING AND GETTING TEST RESULTS");
  m1.setTestResults("0,0,1,1,1,1,0,1,0,0,1,1,1");
  m3.setTestResults("1,1,0,1,0,1,1,1,1,0,1,1,1");
  System.out.println(m1.getAllTestResults());
  System.out.println(m3.getAllTestResults());

  System.out.println();

  System.out.println("TESTING TOSTRING() METHOD");
  System.out.println(m1);  //Movie title: Boo! A Madea Halloween, Number of actors: ##
  m3.addAllActors("nextBechdel_castGender.txt");
  System.out.println(m3);  //Movie title: Sully, Number of actors: ##

  System.out.println();

  System.out.println("TESTING ADDONEACTOR() METHOD");
  m2.addOneActor("'Central Intelligence','Sarah K. Thurber','Beautiful Restaurant
Woman','Supporting','23','Female'");
  System.out.println("EXPECTING SARAH K. THURBER: " + m2.getActors());

  System.out.println();

  System.out.println("TESTING FEMINISTSCORE() METHOD: ");
  System.out.println("Boo Score (expecting 3.0): " + m1.feministScore());
  System.out.println("Sully Score (expecting 2.5): " + m3.feministScore());

  System.out.println();

  System.out.println("TESTING GETORDER() METHOD");
  System.out.println("Movie #1 was created first, expecting 0: " + m1.getOrder());
  System.out.println("Movie #3 was created third, expecting 2: " + m3.getOrder());

  System.out.println();

  System.out.println("TESTING COMPARETO() METHOD: expecting positive number");
  System.out.println(m1.compareTo(m3));
}

}
```

Below is our code for the PriorityQueue Class:

```
package javafoundations;
 /**
 * PriorityQueue.java that represents a priority queue using a LinkedMaxHeap.
 *
 * @author Joyce, Malahim, Candice
 * @version (05/08/2022)
 */
public class PriorityQueue<T extends Comparable<T>> implements Queue<T>
{
  private LinkedMaxHeap<T> heap;
private int count = 0;

/**
 * Constructor. Creates a new priority queue.
 */
public PriorityQueue() {
   heap = new LinkedMaxHeap<T>();
}

/**
 * Adds the specified element to the rear of the queue based on priority.
 *
 * @param element to add to the queue
 */
public void enqueue(T element) {
   heap.add(element);
   count++;
}

/**
 * Removes and returns the element at the front of the queue
 * that has the highest priority.
 *
 * @return element with the highest priority.
 */
public T dequeue() {
   T elt = heap.removeMax();
   count--;
   return elt;
}

/**
 * Returns a reference to the element at the front of the queue
 * with the highest priority without removing it.
 *
 * @return element with the highest priority.
 */
public T first() {
   return heap.getMax();
}

/**
 * Returns true if this priority queue contains no elements and false
 * otherwise.
 *
 * @return true if this priority queue contains no elements and false
 * otherwise.
 */
public boolean isEmpty() {
  return count == 0;
}

/**
 * Returns the number of elements in this priority queue.
 *
 * @return the number of elements in this priority queue.
 */
public int size() {
  return count;
}

/**
 * Returns a string representation of this priority queue.
 *
 * @return a string representation of this priority queue.
 */
public String toString() {
  String result = "";
  PriorityQueue<T> tempQ = new PriorityQueue<T>();

  while(!this.isEmpty()) {
    //dequeue from this collection into the temporary collection
    T movie = this.dequeue();
    tempQ.enqueue(movie);

    result += movie + "\n";
  }

  //put Movies back into this queue from the temporary queue
  while(!tempQ.isEmpty()) {
    T film = tempQ.dequeue();
    this.enqueue(film);
  }
  return result;
}

/**
 * Main method for testing purposes.
```

```
    */
   public static void main(String[] args) {
      PriorityQueue<Double> q = new PriorityQueue<Double>();

      q.enqueue(1.0);
      q.enqueue(2.5);
      q.enqueue(0.0);
      q.enqueue(4.5);
      q.enqueue(4.5);
      q.enqueue(-1.0);

      System.out.println(q);
      System.out.println();

      q.dequeue();
      q.dequeue();

      System.out.println(q);
      System.out.println();

      System.out.println("Queue size: " + q.size());
      System.out.println("Queue should not be empty (expecting false): " + q.isEmpty());
      System.out.println("Current first: " + q.first());

      q.dequeue();
      q.dequeue();
      q.dequeue();
      q.dequeue();

      System.out.println(q);
      System.out.println();

      System.out.println("Queue size: " + q.size());
      System.out.println("Queue should be empty (expecting true): " + q.isEmpty());
   }
}
```

Below is our code for the MovieCollection Class:

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
import javafoundations.PriorityQueue;

/**
 * Write a description of class MovieCollection here.
 *
 * @author Candice, Joyce, Malahim
 * @version 05/02/2022
 */
public class MovieCollection
{
   private LinkedList<Movie> allMovies;
   private LinkedList<Actor> allActors;
   private String testsFileName;
   private String castsFileName;

   /**
    * Constructor for objects of class MovieCollection
    */
   public MovieCollection(String testsFileName, String castsFileName)
   {
      allMovies = new LinkedList<Movie>();
      allActors = new LinkedList<Actor>();
      this.testsFileName = testsFileName;
      this.castsFileName = castsFileName;
   }

   /**
    * Returns all the movies in a LinkedList
    *
    * @return a LinkedList with all the movies, each complete with its title,
    * actors and Bechdel test results.
    */
   public LinkedList<Movie> getMovies() {
      return allMovies;
   }

   /**
    * Returns the titles of all movies in the collection
    *
    * @return a LinkedList with the titles of all the movies
    */
   public LinkedList<String> getMovieTitles() {
      LinkedList<String> allTitles = new LinkedList<String>();
      for(int m=0; m < allMovies.size(); m++) {
        allTitles.add(allMovies.get(m).getTitle());
      }
      return allTitles;
   }

   /**
    * Returns all the Actors in the collection
    *
    * @return a LinkedList with all the Actors,
    * each complete with their name and gender.
    */
   public LinkedList<Actor> getActors() {
      return allActors;
   }

   /**
    * Returns the names of all actors in the collection
    *
    * @return a LinkedList with the names of all actors
    */
   public LinkedList<String> getActorNames() {
      LinkedList<String> allNames = new LinkedList<String>();
      for(int n=0; n < allActors.size(); n++) {
        //for each Actor in this MovieCollection get their name and add to list
        allNames.add(allActors.get(n).getName());
      }
      return allNames;
   }

   /**
    * Returns a String representing this MovieCollection
    *
    * @return a String representation of this collection, including the number
    * of movies and the movies themselves.
    */
   public String toString() {
      String result = "Number of Movies: " + allMovies.size() + "\n";
      for(int i = 0; i < allMovies.size(); i++) {
        result += (allMovies.get(i)).toString();
        result += "\n";
      }
      return result;
   }
```

```
/**
 * Reads the input file, and uses its first column (movie title) to
 * create all movie objects. Adds the included information on the Bachdel
 * test results to each movie.
 */
private void readMovies() {
  try {
    //read from file
    Scanner reader = new Scanner(new File(testsFileName));
    //line of info of actor in this movie
    String lineFromFile = "";
    //skip first line that has titles instead of info of actor
    reader.nextLine();
    while (reader.hasNext()) { //Continue until reach end of input file
      lineFromFile = reader.nextLine();

      //get index of first comma to separate title and testResults
      int index = lineFromFile.indexOf(",");

      //if the line of movie's info has test results
      if(index != -1) {

        String title = lineFromFile.substring(0,index);
        String results = lineFromFile.substring(index+1);

        //create a new movie
        Movie m = new Movie(title);
        //set test results for that movie
        m.setTestResults(results);

        allMovies.add(m);
      }
    }
    reader.close(); // Close the file reader
  } catch (FileNotFoundException ex) {
    System.out.println(ex); // Handle FNF by displaying message
  }
}

//double check: how to add actor objects to allActors
/**
 * Reads the casts for each movie, from input casts file;
 *
 * Assume lines in this file are formatted as followes: "MOVIE","ACTOR","CHARACTER_NAME",
 * "TYPE","BILLING","GENDER"
 *
 * For example: "Trolls","Ricky Dillon","Aspen Heitz","Supporting","18","Male".
 *
 * Notes:
 * 1) each movie will appear in (potentially) many consecutive lines, one line per actor.
 * 2) Each token (title, actor name, etc) appears in double quotes, which need to be
 *    removed as soon as the tokes are read.
 * 3) If a movie does not have any test results, it is ignored and not included in the collection.
 *    (There is actually one such movie)
 */
private void readCasts() {
  //for all movies in this collection with test results
  for(int m = 0; m < allMovies.size(); m++) {
    Movie movie = allMovies.get(m);
    //add all actors in the movie we're reading
    movie.addAllActors(castsFileName);
    Hashtable<Actor, String> h = movie.getAllActors();
    //all the Actors (keys) from hashtable
    Enumeration<Actor> e = h.keys();
    while(e.hasMoreElements()) {
      Actor a = e.nextElement();
      //put all the Actors not in allActors list YET into allActors
      if(!allActors.contains(a)) {
        allActors.add(a);
      }
    }
  }
}

/**
 * Returns a list of all Movies that pass the n-th Bechdel test
 *
 * @param n - integer identifying the n-th test in the list of 12 Bechdel
 * alternatives, starting from zero
 *
 * @return a list of all Movies which have passed the n-th test
 */
public LinkedList<Movie> findAllMoviesPassedTestNum(int n) {
  LinkedList<Movie> passed = new LinkedList<Movie>();
  for(int m = 0; m < allMovies.size(); m++){
    //get the movie's list of test results in the list of all movies in this collection
    Vector results = allMovies.get(m).getAllTestResults();
```

```
      //check that movie's testResults and if it passed the nth test(is equal to zero), add to list
      if(results.get(n).equals("0")) {
        passed.add(allMovies.get(m));
      }
    }
    return passed;
  }

  /**
   * Returns a list of all Movies that passes the Peirce or Landau test.
   *
   * @return LinkedList of all the movies that passes the Peirce or Landau test.
   */
  public LinkedList<Movie> findAllMoviesPassedTestPeirceOrLandau() {
    LinkedList<Movie> passed = new LinkedList<Movie>();
    int peirceIndex = 1;
    int landauIndex = 2;
    for(int m = 0; m < allMovies.size(); m++){
      Vector results = allMovies.get(m).getAllTestResults();
      //check that movie's testResults and if it passed the nth test, add to list
      if(results.get(peirceIndex).equals("0") || results.get(landauIndex).equals("0")) {
        passed.add(allMovies.get(m));
      }
    }
    return passed;
  }

  /**
   * Returns a list of all Movies that passes the White test but not the
   * Rees Davies test.
   *
   * @return LinkedList of all the movies that passes the White test but not the
   * Rees Davies test.
   */
  public LinkedList<Movie> findAllMoviesPassedTestWhiteNotReesDavies() {
    LinkedList<Movie> passed = new LinkedList<Movie>();
    int whiteIndex = 11;
    int reesdaviesIndex = 12;
    for(int m = 0; m < allMovies.size(); m++){
      Vector results = allMovies.get(m).getAllTestResults();
      //check that movie's testResults and if it passed the nth test, add to list
      if(results.get(whiteIndex).equals("0") && results.get(reesdaviesIndex).equals("1")) {
        passed.add(allMovies.get(m));
      }
```

```
        }
        return passed;
    }

    /**
     * Returns a PriorityQueue of movies in the provided data based on their
     * feminist score. That is, if you enqueue all the movies, they will be
     * dequeued in priority order: from most feminist to least feminist.
     *
     * @return PriorityQueue of movies in the provided data based on their
     * feminist score
     */
    public PriorityQueue<Movie> prioritizeMovies() {
        PriorityQueue<Movie> queue = new PriorityQueue<Movie>();
        for (int m = 0; m < allMovies.size(); m++) {
            queue.enqueue(allMovies.get(m));
        }
        return queue;
    }

    /**
     * Main method for testing purposes.
     */
    public static void main(String[] args) {
        System.out.println("***TESTING MOVIECOLLECTION CLASS***");
        System.out.println("Creating a collection of movies; reading movies from file; reading casts from file");
        MovieCollection collection1 = new
MovieCollection("nextBechdel_allTests.txt","nextBechdel_castGender.txt");
        collection1.readMovies();
        collection1.readCasts();

        System.out.println();

        System.out.println("***Testing getActors()***");
        System.out.println("All the actors from the data:\n" + collection1.getActors());

        System.out.println();

        System.out.println("***Testing getActorNames()***");
        System.out.println("All the actors' names from the data:\n" + collection1.getActorNames());

        System.out.println();

        System.out.println("***Testing getMovies()***");
        System.out.println("All the movies from the data:\n" + collection1.getMovies());

        System.out.println();

        System.out.println("***Testing getMovieTitles()***");
        System.out.println("All the movies' titles from the data:\n" + collection1.getMovieTitles());

        System.out.println();

        System.out.println("***Testing findAllMoviesPassedTestNum(n)***");
        System.out.println("Expected 32");
        System.out.println("Number of movies that passed Bechdel Test:  " +
collection1.findAllMoviesPassedTestNum(0).size());
        System.out.println("Expected 0");
        System.out.println("Number of movies that passed Uphold Test:  " +
collection1.findAllMoviesPassedTestNum(10).size());
        System.out.println("Expected 15");
        System.out.println("Number of movies that passed Rees Davis Test:  " +
collection1.findAllMoviesPassedTestNum(12).size());
        System.out.println("Expected 0");
        System.out.println("Number of movies that passed White Test:  " +
collection1.findAllMoviesPassedTestNum(11).size());
        System.out.println("Expected 5");
        System.out.println("Number of movies that passed Waithe Test:  " +
collection1.findAllMoviesPassedTestNum(8).size());
        System.out.println("Expected 21");
        System.out.println("Number of movies that passed Ko Test:  " +
collection1.findAllMoviesPassedTestNum(6).size());
        System.out.println("Expected 0");
        System.out.println("Number of movies that passed VILLAROBOS Test:  " +
collection1.findAllMoviesPassedTestNum(7).size());
        System.out.println("Expected 40");
        System.out.println("Number of movies that passed PEIRCE Test:  " +
collection1.findAllMoviesPassedTestNum(1).size());
        System.out.println("Expected 27");
        System.out.println("Number of movies that passed VILLARREAL Test:  " +
collection1.findAllMoviesPassedTestNum(4).size());
        System.out.println("Expected 17");
        System.out.println("Number of movies that passed LANDAU Test:  " +
collection1.findAllMoviesPassedTestNum(2).size());
        System.out.println("Expected 2 (3 that was supposed to pass did not in testing file)");
        System.out.println("Number of movies that passed HAGEN Test:  " +
collection1.findAllMoviesPassedTestNum(5).size());
        System.out.println("Expected 17");
        System.out.println("Number of movies that passed KOEZE-DOTTLE Test:  " +
collection1.findAllMoviesPassedTestNum(9).size());
        System.out.println("Expected 12");
        System.out.println("Number of movies that passed FELDMAN Test:  " +
collection1.findAllMoviesPassedTestNum(3).size());

        System.out.println();

        System.out.println("***Testing findAllMoviesPassedTestPeirceOrLandau()***");
        System.out.println("Number of movies that passed the Peirce or Landau Test (expecting 47): " +
collection1.findAllMoviesPassedTestPeirceOrLandau().size());
        System.out.println();

        System.out.println("***Testing findAllMoviesPassedTestWhiteNotReesDavies()***");
        System.out.println("Number of movies that passed the White and not ReesDavies Test (expecting 0): " +
collection1.findAllMoviesPassedTestWhiteNotReesDavies().size());
        System.out.println();

        System.out.println();

        System.out.println("***Testing toString()***");
        System.out.println(collection1.toString());

        System.out.println();

        System.out.println("***Testing prioritizeMovies()***");
        System.out.println(collection1.prioritizeMovies());
      }
}
```

VII. Citation

Wezerek, W. H., Rachael Dottle, Ella Koeze, Gus. (2017b, December 21). *Creating the next bechdel Test*. FiveThirtyEight. https://projects.fivethirtyeight.com/next-bechdel/

Vickery, S. (2020, January 13). *Hollywood's gender pay inequity: $1 million per film, UWM researcher finds*. UWM REPORT. Retrieved May 11, 2022, from https://uwm.edu/news/hollywoods-gender-pay-inequity-1-million-per-film-uwm-researcher-finds/#:~:text=According%20to%20new%20research%20conducted,they%20perform%20in%20similar%20roles.

VIII. Collaboration

We did not specifically divide the work because we were able to schedule enough meetings to work on the project together. We worked on pair-programming the project in four 2-3 hour meetings and wrote the report together in two 1-2 hour meetings. Each of us wrote explanations for about 3-4 tests that we chose or didn't choose to define our definition of feminist movies. We each added information to the introduction and conclusion of our report. For the code, we pair-programmed the whole project together. We believe that each of us contributed equally to this project.

We would like to thank Professor Stella Kakavouli and Professor Scott Anderson for helping us debug our code and program parts of the code for this project.