# CSCI 4448 Project 7 Writeup

## Project Details:

- **Project Title:** Sorry! Board Game
- **Group Members:** Brian Noble, Sidhant Puntambekar, Isaac Pyle
- **Code Repository Link:** https://github.com/CSCI-4448-Group/Sorry-Game
- **Demo Video Recording:** https://cuboulder.zoom.us/rec/share/10yiQsFQcSLb8I5VM6rSNtgyJIPMcWYdA-5tYKjwtztuy4tPzl5umiH-5vBkfcsA.mkSpaHz0froW7b1z?startTime=1650925179000
    - Password: iH0.rzSa

## Final State of Submission Details:

- Overall, we are very happy and proud of the Sorry! board game application we developed throughout the course of the semester project. During the development process, we were able to implement nearly all of the promised features and deliverables stated in our project 5 writeup. The features completed during the development period include:
    - Starting up a local instance of the Sorry! game application and landing on the home view
    - Main player/user can start a new game with three other real players
    - Players can draw cards from the game view and have movement handled by the system
    - Having a winner decided by one player getting all of their pawns to the goal square during the course of the game
    - Main player/user can view relevant game statistics and game history on the leaderboard view through a local database connection
    - Main player/user can browse the rules of Sorry! on the rules view
    - Logger is able to log all relevant game interactions that are happening on the board
    - Tracker tracks all game data and summary statistics and saves it to a separate table in a local SQL database
- In terms of features we were not able to set up, we were unable to set up a cloud based database connection, and therefore must run the Sorry! game application through a local MySQL database connection which does limit the functionality of the leaderboard and summary statistics table (however, this functionality is demonstrated through the demo video). Given more time, we would have planned to implement this feature in the future to round off the entire application and backend component. We were also unable to implement the Hibernate ORM for object relational mapping to databases due to a lack of up to database dependencies. Hence, we had to simplify the design with regards to the database and instead opt for a local MySQL database connection instead. We did

make several changes to the class diagram as newly unforeseen data needed to be tracked as well as integrating more design patterns in order to further modularize our game application. For further details on Hibernate, please see the OOAD Process section later in the document.

## Final Class Diagram and Comparison:

- Project 5 Class Diagram: https://lucid.app/lucidchart/3f0f6c9b-ceba-4be0-b844-adc06bc564d6/edit?invitationId=inv_7a6a468c-e7a3-4e99-8feb-9b286ebb16c4
- Project 7 Class Diagram: https://lucid.app/lucidchart/bfa99910-3754-42b5-82d3-778fc05a8db3/edit?invitationId=inv_1964a8bc-dd5a-4e56-9d84-a2896922ed71
- What changed between Project 5 class diagram and Project 7 final class diagram:
    - In terms of changes between the project 5 and project 7 class diagrams, we added additional data attributes between all of the objects for a need to accommodate and track more data in the game state for special card functionalities. This included data relating to total distance traveled by each player's pawns and the number of Sorry! moves the player chose to execute. To this end, we also had to add additional methods (particularly in the command pattern) to accommodate the special functionality of each card move (swapping pawns, splitting the move between a player's current out pawns from home). Furthermore, we updated the class diagram to reflect the new builder and factory patterns we had implemented for constructing the board data structure on the game view. We managed to preserve a lot of the design patterns we initially had set out to implement so we did not change the inherent structure of many of the design pattern sections (command, observer, singleton, object pool) in the UML diagrams from project 5 to project 7. One of these examples includes the use of linked tile objects in a linked list to construct the board on the game view instead of having a dedicated board object. Additionally, we updated the UML diagram to include the DBRunner class which works in concert with the Tracker observer pattern object to update the leaderboard table view and maintain a connection to a local MySQL database instance. More details/comments on pattern usage can be found in the project 5 and project 7 class diagram links above.

## Third-Party code vs. Original code:

- Throughout the project, we estimate that less than 15% of the code in our application came from third party sources. We leveraged the use of the JavaFX framework and associated SceneBuilder as a starting point for our application. As a team however, we completely built objects comprising the board, pawns, players, and resultant game view completely from scratch. Additionally, we implemented all of the relevant special card functionality and deck object (draw card functionality) with the help of the command and object pool design patterns which required consultation of the official Sorry! board game rules and the Java random library. We also made use of third-party code resources that document how to facilitate a connection between a JavaFX application and a local MySQL database instance. This was necessary in order for the lazily instantiated singleton observer tracker object to send data to the database and then having that data be displayed on the leaderboard view. For general pattern guidance, we consulted the *Head First Design Patterns* textbook and the slides from the course. The links to relevant resources (these are labeled in the code) that we leveraged can be found below:

- https://www.geeksforgeeks.org/java-util-random-nextint-java/
- https://www.baeldung.com/java-command-pattern
- https://www.swtestacademy.com/javafx-tutorial/
- https://learning.oreilly.com/library/view/head-first-design/9781492077992/?sso_link=yes&sso_link_from=UniOfColoradoBoulder
- https://www.baeldung.com/java-connect-mysql
- https://www.youtube.com/watch?v=LoiQVoNil9Q&ab_channel=RashidIqbal

## OOAD Process:

- UML Diagramming Before Coding
    - Since our game application was implemented in Java through the JavaFX framework, we were able to maintain the same levels of implementation for the design patterns that we had worked on previously in other projects this semester. Even though the design patterns did not provide total code reuse, they did provide us with experience reuse that we adapted from the earlier Friendly Neighborhood Music Store. We found the process of creating UML diagrams extremely useful upfront as it allowed us to understand which parts of the application were going to need the maximal attention to detail and which ones we could implement right off the bat. Creating UML class diagrams also facilitated our understanding of how the various objects in the game would communicate with each other and how to abstract our designs as much as possible in order to adhere to the object oriented design principles. Overall, we had a positive experience working with UML class diagrams in order to inform our project design at first before sitting down to code the functionality of the application.
- Design Pattern Utilization
    - Throughout the development of the Sorry! board game application, we utilized a number of design patterns in order to isolate the variability in the original system requirements which made working with the objects in the game easier to deal with and handle. Since we had previous project experience to rely on when implementing design patterns in code, we were able to quickly translate that over to the Sorry! game application which helped modularize the overall system we were creating. Throughout the project we utilized the design patterns of singleton (eager logger object, lazy tracker object), observer (logger and tracker objects themselves), model-view-controller (game, leaderboard views), command (translation/delegation from drawn card value to relevant pawn movement), factory (start, goal, gateway tile creation), object pool (card objects, deck objects, player pool object), strategy/template (sorry move, move behavior interface), and

builder (construction of the Sorry! board on the game view). Overall, we had a positive experience working with design patterns throughout our project, and as a team, we were able to integrate many of them into our implementations for the objects comprising the Sorry! game application.

- ORMs and Database Connections
    - We initially had decided to pursue an object relational mapping approach to our integrated database (responsible for keeping track of object states and relevant summary statistics from the game). However, we found that Hibernate (the ORM we were planning on using alongside an SQL database) had a large number of dependencies that were mostly deprecated.
    - We were attempting to use hibernate-core along with some annotations that both required their own dependencies, with specific versions of each requiring other dependencies to have specific matching versions, and when finding out that one would be deprecated it resulted in much time being wasted. As such, after roughly 5 days or so of research into versions that would be compatible and coming up with no solution we decided to abandon the ORM database connection idea and instead set up a simple database connection directly to MySQL through Java in order to send data from the observer pattern tracker object to a simple database table.
    - Overall, the experience working with Hibernate did not pan out the way we intended it to, but this is something that we could potentially investigate in the future as development continues on the project. We briefly looked into hosting a database online using either db4free.net or Heroku. Db4free.net appears to not be suitable for any production environment and can be wiped at any time for any reason. This was not an option, and Heroku had a maximum of 5MB of data that it would store on the database with a free account. Due to these limitations we decided as a proof of concept that the local database would be the best option for our application, and therefore, we implemented a local database connection instead.