Project 7

1. Name of project and names of all team members
Project Title : Exploding Kittens (2 Player) Game
Team Members : Abishek Goutham and Arpita Ambavane

2. Final State of System Statement
• A paragraph on the final state of your system: what features were implemented, what features were not and why, what changed from Project 5 and 6

An Exploding Kitten game is a strategy and luck powered two player game. Players draw cards until somebody draws an Exploding Kitten, at which point they explode and are out of the game. To avoid exploding, they can defuse a kitten OR use powerful action cards to move or avoid the Exploding Kitten. The players have the option to play as many cards as they want during their turn, but they must conclude their turn by drawing a card from the deck of unplayed cards. The game's objective is to avoid drawing the Exploding Kitten card. The last player left alive wins.

- **Features implemented (what not changed)**
  - Taking the Names of the players (input from user).
  - Shuffle the deck of cards at the beginning of the game.
  - Deal seven cards to each player at the start of the game.
  - Allow players to take turns playing a card from their hand.
  - Display the Number of cards left after playing.
  - Display the cards the player has
  - Implement the various card actions, such as exploding kittens, defusing, skipping a turn, and so on.
  - Display the current card on play or played by another player
  - Notifying the turn of the player - For Example: ‘Abishek’s Turn’
  - Allows the player to draw a card from the deck.
  - Alert the user if the option picked is invalid.
  - To place the Exploding Kitten in the deck after playing defuse card - Give options to the player to choose where to place the Exploding Kitten card in the pile.
  - Update the number of cards from players when a favor card is played
  - End the game when only one player is left or when all exploding kittens have been defused.
  - Work on UI functionalities to make our game application more user-friendly.
  - Included a timer functionality for a player to play Defuse card (while exploding)
- **Features implemented (what changed from project 5 and 6)**
  - We didn’t implement an observer pattern in our system.
  - Command Pattern functionality changes
  - Strategy Pattern functionality updates
  - Added Iterator Pattern
  - Added Player and Web Socket Service
  - Added Player and Web Socket Controller

    - Alert the user if the option picked is invalid - this scenario isn't happening in our game as the user can play any card before picking from Deck.
    - Included a timer functionality for a player to play Defuse card (while exploding)
    - When the timer ends, the game automatically ends and declares the other player as winner.
- **Technology/application related features implementation**
    - Created a Spring Boot application
    - Initialized required Classes and interfaces : Game, Game Operations, Player
    - Initialized Controllers and Services for Game, Player and Web Socket
    - Initialized required Classes and interfaces : Hand, Cards
    - Configured API routes to start and get player name of the game
    - Created React Application for User Interface.
    - Created two pages which use the above APIs, i.e. the initial screen and the page where the user enters their names.
    - Configure the third UI page for main game play, i.e. after the start of the game with Player and Card elements.
    - Established a Web Socket connection between React and Spring
    - Included Drag and Drop functionality for the Hand cards, where players can pick or drop the card to the Discard Deck.
    - Implemented design patterns - Command, Factory, State, Strategy, Singleton, and MVC pattern.
- **Issues Encountered in overall application implementation**
    - Difficulty with Drag and Drop functionality of cards
    - Web Socket Implementation for the first time

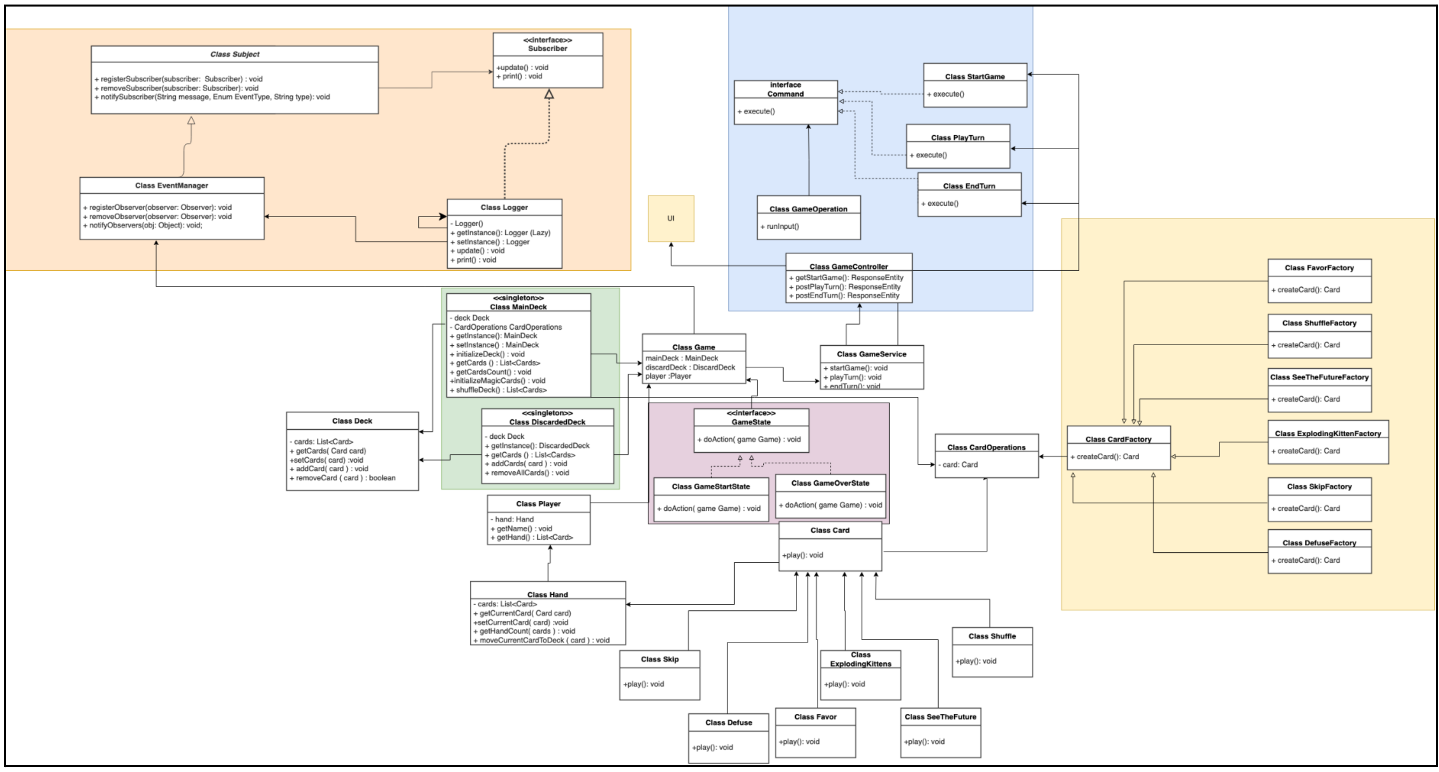
3. Final Class Diagram and Comparison Statement

Class Diagram :

We have added only those elements that we have worked on the past weeks. Design patterns are highlighted.

Project 5 Class Diagram :

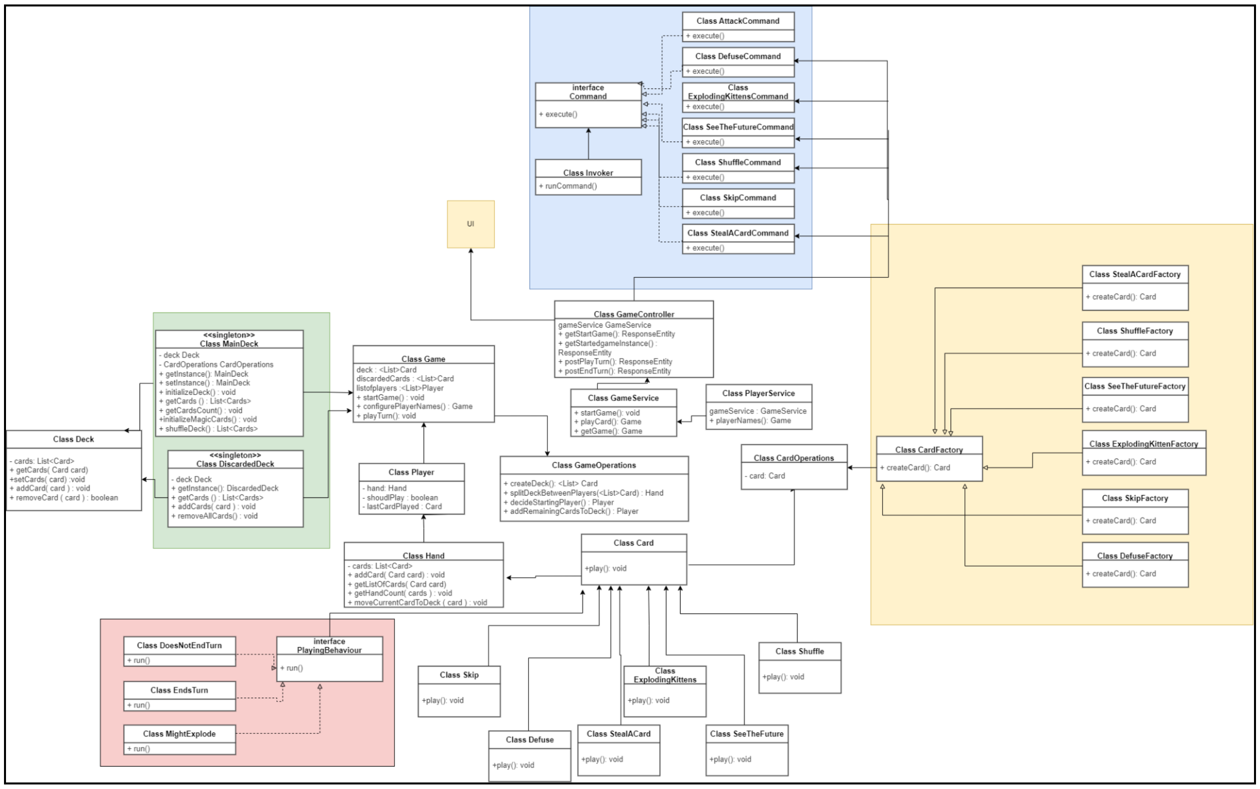
- Design Patterns used
    a. Command Pattern (Blue)
    b. Factory Pattern (Yellow)
    c. State Pattern (Purple)
    d. Observer Pattern (Orange)
    e. Singleton Pattern (Green)



This was the first sketch of the class diagram of the application. We did not implement the observer pattern for logs. We have updated the command pattern functionality. We created GameOperations Class well connected with the Game class for creating decks, splitting players, etc. The state pattern mentioned above seemed a little vague as we had just two game states like start and end state. Hence, we decided to implement another design pattern.

Project 6 Class Diagram :

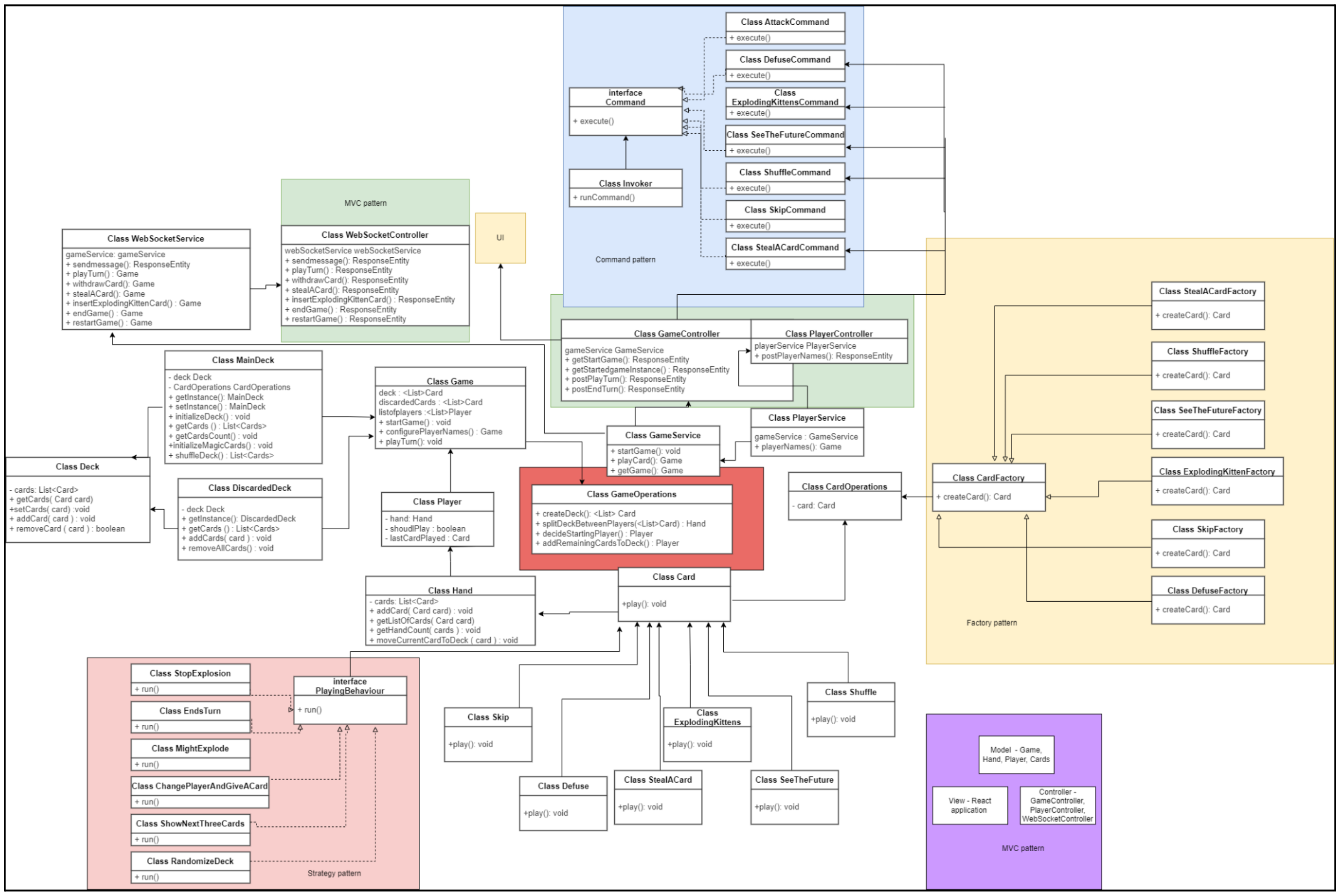
- Design Patterns used
    a. Command Pattern (Blue)
    b. Factory Pattern (Yellow)
    c. Strategy Pattern (Pink)



Here, we updated command pattern functionality to play the commands of the cards actions. Hence CardCommand Classes were created. Game Operations class was connected to Game class. We implemented Strategy pattern for playing strategies behavior i.e. DoesNotEndTurn, EndsTurn, MightExplode. We created Player Class, and services for Game and Player.
We have a Websocket controller and its service but that hasn't been shown in the class diagram.

Project 7 Class Diagram :

- Design Patterns used
    a. Command Pattern (Blue)
    b. Factory Pattern (Yellow)
    c. Strategy Pattern (Pink)
    d. Singleton Pattern (Green) - Autowired Singletons are used to inject a bean of the same type in each controller.
    e. MVC Pattern (Violet) - Model View Controller
    f. Iterator Pattern (Red)
    g. Mediator Pattern - Controller acts as mediator between our Models and Views.



Here, we have used the MVC pattern in our application where Models are Cards, Game, Hand and Player, Views is React application and Controllers are for Game, Player and Websocket. Added Player Controller, Websocket Controller and WebSocket Service in diagram. Updated the Strategy pattern with new classes depicting player behaviors related to action cards. We implemented the Iterator pattern including the usage of in built Iterator. We have implemented Singleton patterns using @autowired tag for wiring Controllers with Services in our Spring Boot Application.

4. Third-Party code vs. Original code Statement

We have not used third-party code so including links referred below resources for different design elements.

Overall Design Patterns -
https://github.com/bethrobson/Head-First-Design-Patterns/tree/master/src/headfirst/designpatterns

Web Socket Implementation - https://www.youtube.com/watch?v=o_IjEDAuo8Y

Drag and Drop functionality -

1. https://react-dnd.github.io/react-dnd/docs/api/hooks-overview
2. https://react-dnd.github.io/react-dnd/docs/api/use-drag
3. https://react-dnd.github.io/react-dnd/docs/api/use-drop

Singleton Pattern -
https://www.waitingforcode.com/spring-framework/singleton-and-prototype-beans-in-spring-framework/read

Strategy Pattern - https://refactoring.guru/design-patterns/strategy

Command Pattern - https://refactoring.guru/design-patterns/command

Factory Pattern - https://refactoring.guru/design-patterns/strategy

5. Statement on the OOAD process for your overall Semester Project
• List three key design process elements or issues (positive or negative) that your team experienced in your analysis and design of the OO semester project

1. Faced difficulties in figuring out which design patterns to use initializing. But we figured it out with time.
2. The project requirements planned in the first iteration were not matching with the timeline such as creating a game against virtual bot. So, we decided to reduce our project scope with respect to project timelines by having a two player game instead. Hence, managing project scope and timelines were important design process elements.
3. We wanted to make our game user interface more friendly or interactive to play for users. So we decided to add a ‘Drag and Drop functionality’ while playing cards instead of using a simple ‘click’ feature.