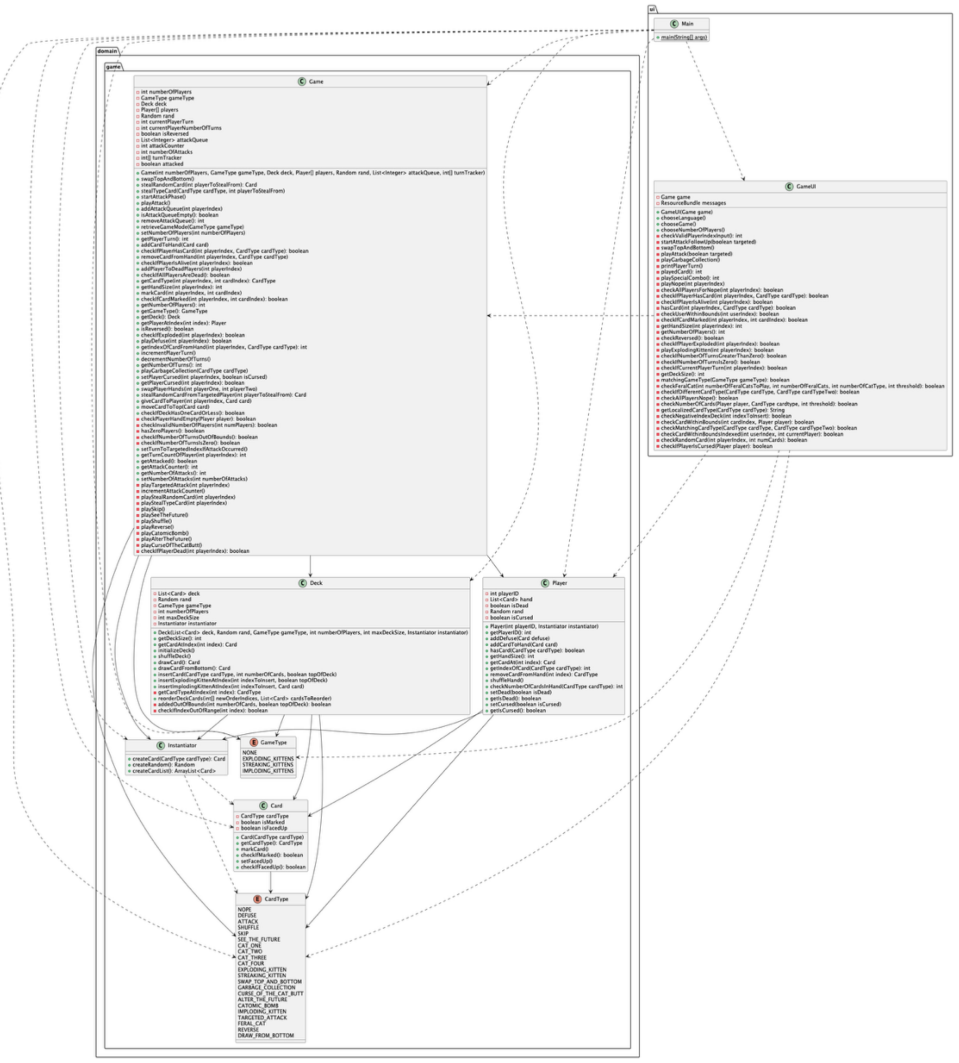
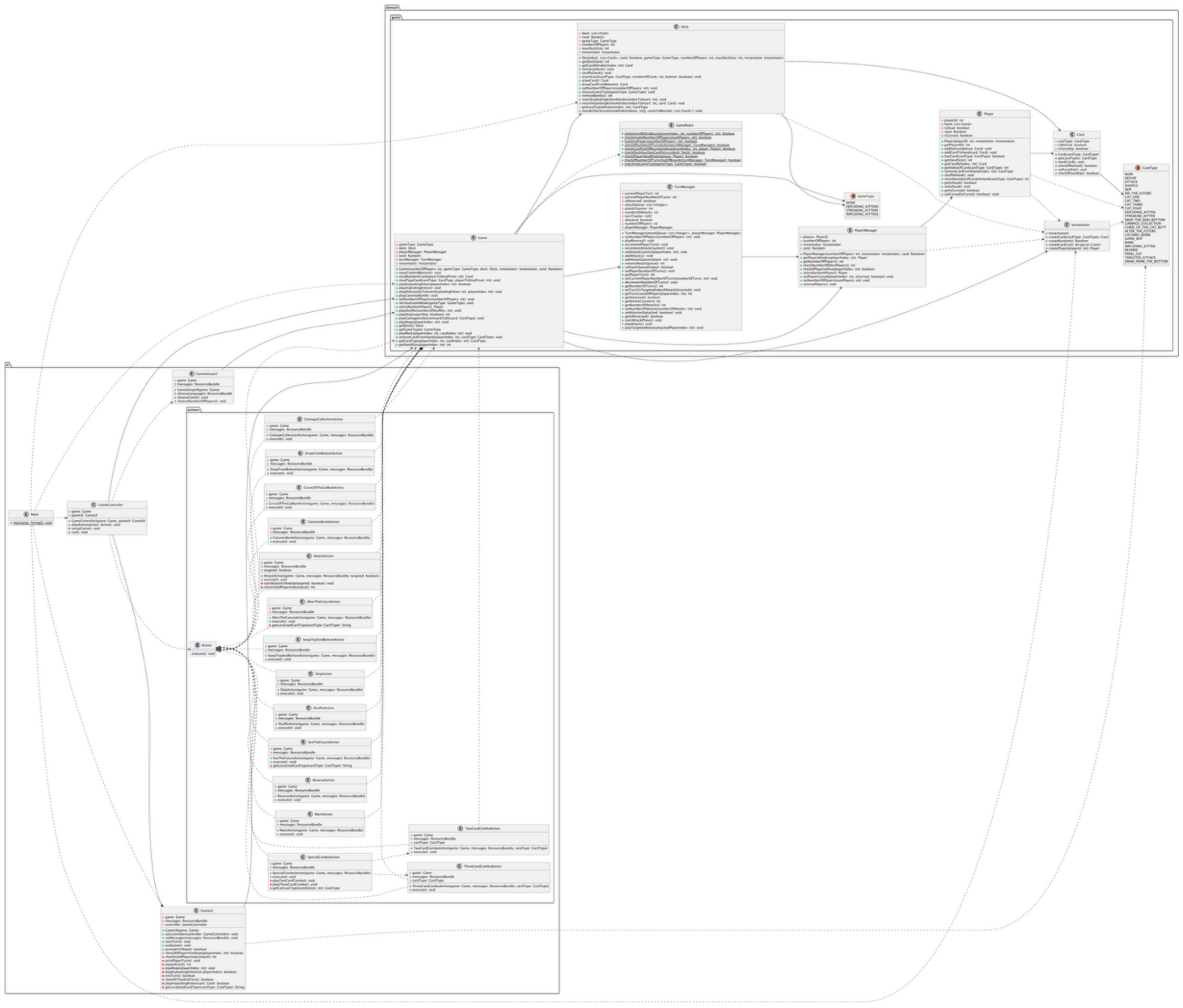
Software Design Document

Design of initial project (Code provided in the end):


ui
Main
main(String[] args)
GameUI
Game game
ResourceBundle messages
GameUI(Game game)
chooseLanguage()
chooseGame()
chooseNumberOfPlayers()
checkValidPlayerIndexInput(): int
startAttackFollowUp(boolean targeted)
swapTopAndBottom()
playAttack(boolean targeted)
playGarbageCollection()
printPlayerTurn()
playedCard(): int
playSpecialCombo(): int
playNope(int playerIndex)
checkAllPlayersForNope(int playerIndex): boolean
checkIfPlayerHasCard(int playerIndex, CardType cardType): boolean
checkIfPlayerIsAlive(int playerIndex): boolean
hasCard(int playerIndex, CardType cardType): boolean
checkUserWithinBounds(int userIndex): boolean
checkIfCardMarked(int playerIndex, int cardIndex): boolean
getHandSize(int playerIndex): int
getNumberOfPlayers(): int
checkReversed(): boolean
checkIfPlayerExploded(int playerIndex): boolean
playExplodingKitten(int playerIndex): boolean
checkIfNumberOfTurnsGreaterThanZero(): boolean
checkIfNumberOfTurnsIsZero(): boolean
checkIfCurrentPlayerTurn(int playerIndex): boolean
getDeckSize(): int
matchingGameType(GameType gameType): boolean
checkFeralCat(int numberOfFeralCatsToPlay, int numberOfFeralCats, int numberOfCatType, int threshold): boolean
checkIfDifferentCardType(CardType cardType, CardType cardTypeTwo): boolean
checkAllPlayersNope(): boolean
checkNumberOfCards(Player player, CardType cardtype, int threshold): boolean
getLocalizedCardType(CardType cardType): String
checkNegativeIndexDeck(int indexToInsert): boolean
checkCardWithinBounds(int cardIndex, Player player): boolean
checkMatchingCardType(CardType cardType, CardType cardTypeTwo): boolean
checkCardWithinBoundsIndexed(int userIndex, int currentPlayer): boolean
checkRandomCard(int playerIndex, int numCards): boolean
checkIfPlayerIsCursed(Player player): boolean
domain
game
Game
int numberOfPlayers
GameType gameType
Deck deck
Player[] players
Random rand
int currentPlayerTurn
int currentPlayerNumberOfTurns
boolean isReversed
List<Integer> attackQueue
int attackCounter
int numberOfAttacks
int[] turnTracker
boolean attacked
Game(int numberOfPlayers, GameType gameType, Deck deck, Player[] players, Random rand, List<Integer> attackQueue, int[] turnTracker)
swapTopAndBottom()
stealRandomCard(int playerToStealFrom): Card
stealTypeCard(CardType cardType, int playerToStealFrom)
startAttackPhase()
playAttack()
addAttackQueue(int playerIndex)
isAttackQueueEmpty(): boolean
removeAttackQueue(): int
retrieveGameMode(GameType gameType)
setNumberOfPlayers(int numberOfPlayers)
getPlayerTurn(): int
addCardToHand(Card card)
checkIfPlayerHasCard(int playerIndex, CardType cardType): boolean
removeCardFromHand(int playerIndex, CardType cardType)
checkIfPlayerIsAlive(int playerIndex): boolean
addPlayerToDeadPlayers(int playerIndex)
checkIfAllPlayersAreDead(): boolean
getCardType(int playerIndex, int cardIndex): CardType
getHandSize(int playerIndex): int
markCard(int playerIndex, int cardIndex)
checkIfCardMarked(int playerIndex, int cardIndex): boolean
getNumberOfPlayers(): int
getGameType(): GameType
getDeck(): Deck
getPlayerAtIndex(int index): Player
isReversed(): boolean
checkIfExploded(int playerIndex): boolean
playDefuse(int playerIndex): boolean
getIndexOfCardFromHand(int playerIndex, CardType cardType): int
incrementPlayerTurn()
decrementNumberOfTurns()
getNumberOfTurns(): int
playGarbageCollection(CardType cardType)
setPlayerCursed(int playerIndex, boolean isCursed)
getPlayerCursed(int playerIndex): boolean
swapPlayerHands(int playerOne, int playerTwo)
stealRandomCardFromTargetedPlayer(int playerToStealFrom): Card
giveCardToPlayer(int playerIndex, Card card)
moveCardToTop(Card card)
checkIfDeckHasOneCardOrLess(): boolean
checkPlayerHandEmpty(Player player): boolean
checkInvalidNumberOfPlayers(int numPlayers): boolean
hasZeroPlayers(): boolean
checkIfNumberOfTurnsOutOfBounds(): boolean
checkIfNumberOfTurnsIsZero(): boolean
setTurnToTargetedIndexIfAttackOccurred()
getTurnCountOfPlayer(int playerIndex): int
getAttacked(): boolean
getAttackCounter(): int
getNumberOfAttacks(): int
setNumberOfAttacks(int numberOfAttacks)
playTargetedAttack(int playerIndex)
incrementAttackCounter()
playStealRandomCard(int playerIndex)
playStealTypeCard(int playerIndex)
playSkip()
playSeeTheFuture()
playShuffle()
playReverse()
playCatomicBomb()
playAlterTheFuture()
playCurseOfTheCatButt()
checkIfPlayerDead(int playerIndex): boolean
Deck
List<Card> deck
Random rand
GameType gameType
int numberOfPlayers
int maxDeckSize
Instantiator instantiator
Deck(List<Card> deck, Random rand, GameType gameType, int numberOfPlayers, int maxDeckSize, Instantiator instantiator)
getDeckSize(): int
getCardAtIndex(int index): Card
initializeDeck()
shuffleDeck()
drawCard(): Card
drawCardFromBottom(): Card
insertCard(CardType cardType, int numberOfCards, boolean topOfDeck)
insertExplodingKittenAtIndex(int indexToInsert, boolean topOfDeck)
insertImplodingKittenAtIndex(int indexToInsert, Card card)
getCardTypeAtIndex(int index): CardType
reorderDeckCards(int[] newOrderIndices, List<Card> cardsToReorder)
addedOutOfBounds(int numberOfCards, boolean topOfDeck): boolean
checkIfIndexOutOfRange(int index): boolean
Player
int playerID
List<Card> hand
boolean isDead
Random rand
boolean isCursed
Player(int playerID, Instantiator instantiator)
getPlayerID(): int
addDefuse(Card defuse)
addCardToHand(Card card)
hasCard(CardType cardType): boolean
getHandSize(): int
getCardAt(int index): Card
getIndexOfCard(CardType cardType): int
removeCardFromHand(int index): CardType
shuffleHand()
checkNumberOfCardsInHand(CardType cardType): int
setDead(boolean isDead)
getIsDead(): boolean
setCursed(boolean isCursed)
getIsCursed(): boolean
Instantiator
createCard(CardType cardType): Card
createRandom(): Random
createCardList(): ArrayList<Card>
GameType
NONE
EXPLODING_KITTENS
STREAKING_KITTENS
IMPLODING_KITTENS
Card
CardType cardType
boolean isMarked
boolean isFacedUp
Card(CardType cardType)
getCardType(): CardType
markCard()
checkIfMarked(): boolean
setFacedUp()
checkIfFacedUp(): boolean
CardType
NOPE
DEFUSE
ATTACK
SHUFFLE
SKIP
SEE_THE_FUTURE
CAT_ONE
CAT_TWO
CAT_THREE
CAT_FOUR
EXPLODING_KITTEN
STREAKING_KITTEN
SWAP_TOP_AND_BOTTOM
GARBAGE_COLLECTION
CURSE_OF_THE_CAT_BUTT
ALTER_THE_FUTURE
CATOMIC_BOMB
IMPLODING_KITTEN
TARGETED_ATTACK
FERAL_CAT
REVERSE
DRAW_FROM_BOTTOM

Functional and flexible design of the product (Code provided in the end):



A list of major changes made to the design, and the analysis of why the changed was needed:

The original project likely followed a monolithic design, common for simpler applications, with most of the code residing in just a few files.

```
src/
 └──── main/
  └──── java/
  ├──── domain/
              ├──── Game.java// Likely contained all game logic, rules, and state(huge file)
              ├────.......
  └──── ui/
              ├────GameUI.java// Handled user input, display, and the process of UI(huge file)
              ├────........
```

Current Refactored Structure

The refactored project exhibits a clear separation of concerns, with a well-defined package structure that distributes responsibilities across multiple, specialized classes.

```
src/
 └──── main/
  └──── java/
  ├──── domain/
  │   └──── game/
  │   ├──── Card.java
  │   ├──── CardType.java
  │   ├──── Deck.java
  │   ├──── Game.java
  │   ├──── GameRules.java  // new added
  │   ├──── GameType.java
  │  ├──── Instantiator.java
  │  ├──── Player.java
  │  ├──── PlayerManager.java                    // new added
  │  └──── TurnManager.java            // new added
  └──── ui/
  ├──── actions/
  │  ├──── Action.java                  // new added
  │  ├──── AlterTheFutureAction.java  // new added
  │  ├──── AttackAction.java            // new added
  │  ├──── // ... and 10+ other action classes     // new added
  │  └──── TwoCardComboAction.java              // new added
  ├──── GameController.java            // new added
  ├──── GameSetupUI.java   // new added
  ├──── GameUI.java
  └──── Main.java
```

There are the major changes I have made to improve the project's maintainability, scalability, and testability:

1. Adoption of the Model-View-Controller (MVC) Pattern

Change Description: The most fundamental change was the introduction of the MVC architectural pattern. The single default package was split into domain.game (Model) and ui (View/Controller).

- Model (domain.game): This package is the source of truth. It contains all the core game entities (Card, Player, Deck), state (Game), and business logic (GameRules). It knows nothing about how the game is presented to the user. It is a pure, self-contained representation of the game world.
- View (GameUI, GameSetupUI): This is the presentational layer. Its sole responsibility is to render the state of the Model for the user and capture

user input. It handles all System.out.println calls and reads from System.in. It is "dumb" in that it does not contain any game logic; it only displays what it's told and reports user actions to the Controller.

- Controller (GameController, ui.actions): This is the orchestrator. It acts as the intermediary between the Model and the View. It takes user input from the View (e.g., "the user wants to play an Attack card"), translates it into a command, and instructs the Model to update its state accordingly.

Why it was Needed: The original monolithic design tightly coupled the game logic with the user interface. This makes the code brittle and difficult to maintain. For example, to test the game logic, one would have to run the entire application. To change the UI from a console application to a graphical one would require rewriting the entire codebase. MVC decouples these layers, providing immense benefits:

1. The entire domain.game package can be unit-tested independently of any UI, ensuring the core game rules are correct.
2. The console-based GameUI could be swapped out for a graphical JavaFX or web-based UI without changing a single line of code in the domain.game package.
3. The code is far easier to understand and debug because responsibilities are clearly defined and isolated.

Specific File Operations:

- All logic pertaining to the state of the game was consolidated within the classes of the domain.game package.
- Game.java was refactored to remove any and all System.out or Scanner calls. It now only contains methods that manipulate the internal game state (e.g., drawCard(), playCard()).
- All user-facing text, menus, and input reading were moved into GameUI.java and GameSetupUI.java.

2. Implementation of the Command Pattern for Player Actions

Change Description: Every distinct player action was encapsulated into its own class within the ui.actions package, implementing a common Action interface. This pattern decouples the object that invokes an operation (the GameController) from the object that knows how to perform it (the specific Action class).

Why it was Needed: A large conditional block for actions violates the Open/Closed Principle and creates high coupling. The Command pattern solves this by turning a request into a stand-alone object. This change provides several key advantages:

1. Adding a new card and its corresponding action is now trivial. A developer creates a new class in the ui.actions package, implements the Action interface, and adds it as an option in the GameUI. No other part of the system needs to be modified, drastically reducing the risk of introducing regression bugs.
2. The GameController is completely decoupled from the specific actions. It doesn't know what a ShuffleAction or AttackAction does; its only responsibility is to execute any object that conforms to the Action interface. This simplifies the controller's logic immensely.
3. Action objects can be parameterized. For instance, the ThreeCardComboAction is initialized with the target player and the specific card to steal, information it carries with it until it is executed.

Specific File Operations:

- The creation of the ui.actions package established a clear home for all command objects.
- The Action.java interface was defined with a single method, execute(), ensuring all commands have a uniform entry point.
- The lifecycle of a command is now clear: GameUI creates the specific action instance based on user input (e.g., new AttackAction(game, targetedPlayer)), this instance is passed to the GameController, and the controller calls execute(). This flow is consistent for all actions.

3. Introduction of a Central Game Controller

Change Description: A GameController.java class was introduced to serve as the application's central nervous system. It exclusively owns the main game loop and is the only component that should be aware of both the Model (Game) and the View (GameUI).

Why it was Needed: The controller centralizes all program flow control, preventing the code where UI elements might directly alter game state, leading to unpredictable behavior. It creates a clear, unidirectional data flow that is easy to trace and debug. The controller's existence cleans up the Main class, whose only job now is to instantiate the MVC components and start the controller, a hallmark of a well-architected application.

Specific File Operations:

The main game loop (while (!isGameOver)) was moved into GameController.run(). This method now represents the entire lifecycle of a game session from start to finish.

A typical turn from the controller's perspective is:

1. Query the TurnManager to get the current player.
2. Invoke gameUI.startTurn(currentPlayer) to render the view.
3. Receive a chosen Action object back from the UI.
4. Invoke nopeMechanism.run(action) to see if the action is canceled.
5. If not canceled, call action.execute().
6. Check the PlayerManager to see if the game is over.
7. Tell the TurnManager to advance to the next player.

This sequence demonstrates how the controller perfectly mediates all interactions, ensuring the Model and View remain independent.

4. Delegation of Responsibility with Manager Classes

Change Description: Logic for complex subdomains—namely player and turn management—was extracted from the Game class into PlayerManager.java and TurnManager.java. This prevents the Game class from becoming a "God Object," an anti-pattern where one class knows and does too much.

Why it was Needed: This change is a direct application of the Single Responsibility Principle (SRP). By delegating, we create classes that are smaller, more cohesive, and easier to test and maintain.

1. These manager classes properly encapsulate their data. For instance, no other class can get direct access to the List<Player> inside PlayerManager. Instead, they must use its public methods like getActivePlayers(). This protects the integrity of the player list.
2. The Game class is now a high-level coordinator. It doesn't need to know the complex details of how to reverse turn order or handle an Attack card's double-turn effect; it simply tells the TurnManager what happened, and the TurnManager adjusts its internal state accordingly.

Specific File Operations:

- PlayerManager.java was created to be the sole authority on player state. All methods related to adding, eliminating, or querying players now reside here.
- TurnManager.java was created to encapsulate all turn-sequencing logic. It handles the currentPlayerIndex, the direction of play (isReversed), and the queue of turns to be taken (turnsToTake). This successfully isolates some of the most complex logic in the game into a single, focused class.
- The Game.java class was simplified; its methods now delegate these specific tasks, such as game.getPlayerManager().eliminatePlayer(p), making its own responsibilities clearer.

These four transformations were major changes I have made for the existing code. In fact, I also made some minor changes that I won't elaborate on here. While the game's functionality remains the same for the end-user, the underlying structure has been fundamentally improved. I really like the journey of refactoring and processing the existing java code. In the end, I'd like to thank our professor and TAs for their guidance and insights throughout the quarter; this course taught me the rigorous software-design principles, and I've learned a great deal.

Design of initial project Code:

```
@startuml
package ui {
    class GameUI {
        - Game game
        - ResourceBundle messages
        + GameUI(Game game)
        + chooseLanguage()
        + chooseGame()
        + chooseNumberOfPlayers()
        - checkValidPlayerIndexInput(): int
        - startAttackFollowUp(boolean targeted)
        - swapTopAndBottom()
        - playAttack(boolean targeted)
        - playGarbageCollection()
        - printPlayerTurn()
        - playedCard(): int
        - playSpecialCombo(): int
        - playNope(int playerIndex)
        - checkAllPlayersForNope(int playerIndex): boolean
        - checkIfPlayerHasCard(int playerIndex, CardType cardType): boolean
        - checkIfPlayerIsAlive(int playerIndex): boolean
        - hasCard(int playerIndex, CardType cardType): boolean
        - checkUserWithinBounds(int userIndex): boolean
        - checkIfCardMarked(int playerIndex, int cardIndex): boolean
        - getHandSize(int playerIndex): int
        - getNumberOfPlayers(): int
        - checkReversed(): boolean
        - checkIfPlayerExploded(int playerIndex): boolean
        - playExplodingKitten(int playerIndex): boolean
        - checkIfNumberOfTurnsGreaterThanZero(): boolean
        - checkIfNumberOfTurnsIsZero(): boolean
        - checkIfCurrentPlayerTurn(int playerIndex): boolean
        - getDeckSize(): int
        - matchingGameType(GameType gameType): boolean
        - checkFeralCat(int numberOfFeralCatsToPlay, int numberOfFeralCats,
int numberOfCatType, int threshold): boolean
        - checkIfDifferentCardType(CardType cardType, CardType cardTypeTwo):
boolean
        - checkAllPlayersNope(): boolean
        - checkNumberOfCards(Player player, CardType cardtype, int
threshold): boolean
        - getLocalizedCardType(CardType cardType): String
        - checkNegativeIndexDeck(int indexToInsert): boolean
        - checkCardWithinBounds(int cardIndex, Player player): boolean
        - checkMatchingCardType(CardType cardType, CardType cardTypeTwo):
boolean
        - checkCardWithinBoundsIndexed(int userIndex, int currentPlayer):
boolean
        - checkRandomCard(int playerIndex, int numCards): boolean
        - checkIfPlayerIsCursed(Player player): boolean
    }

    class Main {
        + {static} main(String[] args)
    }
}

package domain.game {
    class Card {
        - CardType cardType
        - boolean isMarked
        - boolean isFacedUp
        + Card(CardType cardType)
        + getCardType(): CardType
        + markCard()
        + checkIfMarked(): boolean
        + setFacedUp()
        + checkIfFacedUp(): boolean
    }

    enum CardType {
        NOPE
        DEFUSE
        ATTACK
        SHUFFLE
        SKIP
        SEE_THE_FUTURE
        CAT_ONE
        CAT_TWO
        CAT_THREE
        CAT_FOUR
        EXPLODING_KITTEN
        STREAKING_KITTEN
        SWAP_TOP_AND_BOTTOM
        GARBAGE_COLLECTION
        CURSE_OF_THE_CAT_BUTT
        ALTER_THE_FUTURE
        CATOMIC_BOMB
        IMPLODING_KITTEN
        TARGETED_ATTACK
        FERAL_CAT
        REVERSE
        DRAW_FROM_BOTTOM
    }

    class Deck {
        - List<Card> deck
        - Random rand
        - GameType gameType
        - int numberOfPlayers
        - int maxDeckSize
        - Instantiator instantiator
        + Deck(List<Card> deck, Random rand, GameType gameType, int
numberOfPlayers, int maxDeckSize, Instantiator instantiator)
        + getDeckSize(): int
        + getCardAtIndex(int index): Card
        + initializeDeck()
        + shuffleDeck()
        + drawCard(): Card
        + drawCardFromBottom(): Card
        + insertCard(CardType cardType, int numberOfCards, boolean topOfDeck)
        + insertExplodingKittenAtIndex(int indexToInsert, boolean topOfDeck)
        + insertImplodingKittenAtIndex(int indexToInsert, Card card)
        - getCardTypeAtIndex(int index): CardType
        + reorderDeckCards(int[] newOrderIndices, List<Card> cardsToReorder)
        - addedOutOfBounds(int numberOfCards, boolean topOfDeck): boolean
        - checkIfIndexOutOfRange(int index): boolean
    }

    class Game {
        - int numberOfPlayers
        - GameType gameType
        - Deck deck
        - Player[] players
        - Random rand
        - int currentPlayerTurn
        - int currentPlayerNumberOfTurns
        - boolean isReversed
        - List<Integer> attackQueue
        - int attackCounter
        - int numberOfAttacks
        - int[] turnTracker
        - boolean attacked
        + Game(int numberOfPlayers, GameType gameType, Deck deck, Player[]
players, Random rand, List<Integer> attackQueue, int[] turnTracker)
        + swapTopAndBottom()
        + stealRandomCard(int playerToStealFrom): Card
        + stealTypeCard(CardType cardType, int playerToStealFrom)
        + startAttackPhase()
        + playAttack()
        + addAttackQueue(int playerIndex)
        + isAttackQueueEmpty(): boolean
        + removeAttackQueue(): int
        + retrieveGameMode(GameType gameType)
        + setNumberOfPlayers(int numberOfPlayers)
        + getPlayerTurn(): int
        + addCardToHand(Card card)
        + checkIfPlayerHasCard(int playerIndex, CardType cardType): boolean
        + removeCardFromHand(int playerIndex, CardType cardType)
        + checkIfPlayerIsAlive(int playerIndex): boolean
        + addPlayerToDeadPlayers(int playerIndex)
        + checkIfAllPlayersAreDead(): boolean
        + getCardType(int playerIndex, int cardIndex): CardType
        + getHandSize(int playerIndex): int
        + markCard(int playerIndex, int cardIndex)
        + checkIfCardMarked(int playerIndex, int cardIndex): boolean
        + getNumberOfPlayers(): int
        + getGameType(): GameType
        + getDeck(): Deck
        + getPlayerAtIndex(int index): Player
        + isReversed(): boolean
        + checkIfExploded(int playerIndex): boolean
        + playDefuse(int playerIndex): boolean
        + getIndexOfCardFromHand(int playerIndex, CardType cardType): int
        + incrementPlayerTurn()
        + decrementNumberOfTurns()
        + getNumberOfTurns(): int
        + playGarbageCollection(CardType cardType)
        + setPlayerCursed(int playerIndex, boolean isCursed)
        + getPlayerCursed(int playerIndex): boolean
        + swapPlayerHands(int playerOne, int playerTwo)
        + stealRandomCardFromTargetedPlayer(int playerToStealFrom): Card
        + giveCardToPlayer(int playerIndex, Card card)
        + moveCardToTop(Card card)
        + checkIfDeckHasOneCardOrLess(): boolean
        - checkPlayerHandEmpty(Player player): boolean
        - checkInvalidNumberOfPlayers(int numPlayers): boolean
        - hasZeroPlayers(): boolean
        - checkIfNumberOfTurnsOutOfBounds(): boolean
        - checkIfNumberOfTurnsIsZero(): boolean
        + setTurnToTargetedIndexIfAttackOccurred()
        + getTurnCountOfPlayer(int playerIndex): int
        + getAttacked(): boolean
        + getAttackCounter(): int
        + getNumberOfAttacks(): int
        + setNumberOfAttacks(int numberOfAttacks)
        - playTargetedAttack(int playerIndex)
        - incrementAttackCounter()
        - playStealRandomCard(int playerIndex)
        - playStealTypeCard(int playerIndex)
        - playSkip()
        - playSeeTheFuture()
        - playShuffle()
        - playReverse()
        - playCatomicBomb()
        - playAlterTheFuture()
        - playCurseOfTheCatButt()
        - checkIfPlayerDead(int playerIndex): boolean
    }

    enum GameType {
        NONE
        EXPLODING_KITTENS
        STREAKING_KITTENS
        IMPLODING_KITTENS
    }

    class Instantiator {
        + createCard(CardType cardType): Card
        + createRandom(): Random
        + createCardList(): ArrayList<Card>
    }

    class Player {
        - int playerID
```

```
        - List<Card> hand
        - boolean isDead
        - Random rand
        - boolean isCursed
        + Player(int playerID, Instantiator instantiator)
        + getPlayerID(): int
        + addDefuse(Card defuse)
        + addCardToHand(Card card)
        + hasCard(CardType cardType): boolean
        + getHandSize(): int
        + getCardAt(int index): Card
        + getIndexOfCard(CardType cardType): int
        + removeCardFromHand(int index): CardType
        + shuffleHand()
        + checkNumberOfCardsInHand(CardType cardType): int
        + setDead(boolean isDead)
        + getIsDead(): boolean
        + setCursed(boolean isCursed)
        + getIsCursed(): boolean
    }
}

' Relationships
GameUI ..> Game
GameUI ..> CardType
GameUI ..> Player
GameUI ..> GameType

Main ..> GameUI
Main ..> Game
Main ..> Deck
Main ..> Player
Main ..> Instantiator
Main ..> Card
Main ..> CardType
Main ..> GameType

Game --> Deck
Game --> Player
Game --> GameType
Game --> CardType
Game --> Instantiator

Deck --> Card
Deck --> Instantiator
Deck --> CardType
Deck --> GameType

Card --> CardType

Player --> Card
Player --> Instantiator
Player --> CardType

Instantiator ..> Card
Instantiator ..> CardType
@enduml
```

Functional and flexible design of the product code:

```
@startuml
left to right direction
scale 3000 * 3000
package "domain.game" {

  class Card {
    - cardType: CardType
    - isMarked: boolean
    - isFacedUp: boolean
    + Card(cardType: CardType)
    + getCardType(): CardType
    + markCard(): void
    + checkIfMarked(): boolean
    + setFacedUp(): void
    + checkIfFacedUp(): boolean
  }

  enum CardType {
    NOPE
    DEFUSE
    ATTACK
    SHUFFLE
    SKIP
    SEE_THE_FUTURE
    CAT_ONE
    CAT_TWO
    CAT_THREE
    CAT_FOUR
    EXPLODING_KITTEN
    STREAKING_KITTEN
    SWAP_TOP_AND_BOTTOM
    GARBAGE_COLLECTION
    CURSE_OF_THE_CAT_BUTT
    ALTER_THE_FUTURE
    CATOMIC_BOMB
    SUPER_SKIP
    MARK
    IMPLODING_KITTEN
    REVERSE
    FERAL_CAT
    TARGETED_ATTACK
    DRAW_FROM_THE_BOTTOM
  }

  class Deck {
    - deck: List<Card>
    - rand: Random
    - gameType: GameType
    - numberOfPlayers: int
    - maxDeckSize: int
    - instantiator: Instantiator
    + Deck(deck: List<Card>, rand: Random, gameType: GameType,
numberOfPlayers: int, maxDeckSize: int, instantiator: Instantiator)
    + getDeckSize(): int
    + getCardAtIndex(index: int): Card
    + initializeDeck(): void
    + shuffleDeck(): void
    + insertCard(cardType: CardType, numberOfCards: int, bottom: boolean):
void
    + drawCard(): Card
    + drawCardFromBottom(): Card
    + setNumberOfPlayers(numberOfPlayers: int): void
    + chooseGameType(gameType: GameType): void
    + removeBombs(): int
    + insertExplodingKittenAtIndex(indexToInsert: int): void
    + insertImplodingKittenAtIndex(indexToInsert: int, card: Card): void
    # getCardTypeAtIndex(index: int): CardType
    + reorderDeckCards(newOrderIndices: int[], cardsToReorder: List<Card>):
void
  }

  class Game {
    - gameType: GameType
    - deck: Deck
    - playerManager: PlayerManager
    - rand: Random
    - turnManager: TurnManager
    - instantiator: Instantiator
    + Game(numberOfPlayers: int, gameType: GameType, deck: Deck,
instantiator: Instantiator, rand: Random)
    + swapTopAndBottom(): void
    + stealRandomCard(playerToStealFrom: int): Card
    + stealTypeCard(cardType: CardType, playerToStealFrom: int): void
    + playExplodingKitten(playerIndex: int): boolean
    + playImplodingKitten(): void
    + playDefuse(idxToInsertExplodingKitten: int, playerIndex: int): void
    + playCatomicBomb(): void
    + setNumberOfPlayers(numberOfPlayers: int): void
    + retrieveGameMode(gameType: GameType): void
    + selectRandomPlayer(): Player
    + playShuffle(numberOfShuffles: int): void
    + playSkip(superSkip: boolean): int
    + playGarbageCollection(cardToDiscard: CardType): void
    + playNope(playerIndex: int): void
    + getDeck(): Deck
    + getGameType(): GameType
    + playMark(playerIndex: int, cardIndex: int): void
    + removeCardFromHand(playerIndex: int, cardType: CardType): void
    + getCardType(playerIndex: int, cardIndex: int): CardType
    + getHandSize(playerIndex: int): int
  }

  class GameRules {
    + {static} checkUserWithinBounds(userIndex: int, numberOfPlayers: int):
boolean
    + {static} checkInvalidNumberOfPlayers(numPlayers: int): boolean
    + {static} hasZeroPlayers(numberOfPlayers: int): boolean
    + {static} checkIfNumberOfTurnsIsZero(turnManager: TurnManager): boolean
    + {static} checkCardOutOfBoundsIndexed(cardIndex: int, player: Player):
boolean
    + {static} checkDeckHasOneCardOrLess(deck: Deck): boolean
    + {static} checkPlayerHandEmpty(player: Player): boolean
    + {static} checkIfNumberOfTurnsOutOfBounds(turnManager: TurnManager):
boolean
    + {static} matchingGameType(gameType: GameType): boolean
  }

  enum GameType {
    NONE
    EXPLODING_KITTENS
    STREAKING_KITTENS
    IMPLODING_KITTENS
  }

  class Instantiator {
    + Instantiator()
    + createCard(cardType: CardType): Card
    + createRandom(): Random
    + createCardList(): ArrayList<Card>
    + createPlayer(playerId: int): Player
  }

  class Player {
    - playerID: int
    - hand: List<Card>
    - isDead: boolean
    - rand: Random
    - isCursed: boolean
    + Player(playerID: int, instantiator: Instantiator)
    + getPlayerID(): int
    + addDefuse(defuse: Card): void
    + addCardToHand(card: Card): void
    + hasCard(cardType: CardType): boolean
    + getHandSize(): int
    + getCardAt(index: int): Card
    + getIndexOfCard(cardType: CardType): int
    + removeCardFromHand(index: int): CardType
    + shuffleHand(): void
    + checkNumberOfCardsInHand(cardType: CardType): int
    + getIsDead(): boolean
    + setIsDead(): void
    + getIsCursed(): boolean
    + setCursed(isCursed: boolean): void
  }

  class PlayerManager {
    - players: Player[]
    - numberOfPlayers: int
    - instantiator: Instantiator
    - rand: Random
    + PlayerManager(numberOfPlayers: int, instantiator: Instantiator, rand:
Random)
    + getPlayerAtIndex(playerIndex: int): Player
    + getNumberOfPlayers(): int
    + checkNumberOfAlivePlayers(): int
    + checkIfPlayerIsAlive(playerIndex: int): boolean
    + selectRandomPlayer(): Player
    + setPlayerCursed(playerIndex: int, isCursed: boolean): void
    + setNumberOfPlayers(numPlayers: int): void
    + reversePlayers(): void
  }

  class TurnManager {
    - currentPlayerTurn: int
    - currentPlayerNumberOfTurns: int
    - isReversed: boolean
    - attackQueue: List<Integer>
    - attackCounter: int
    - numberOfAttacks: int
    - turnTracker: int[]
    - attacked: boolean
    - numberOfPlayers: int
    - playerManager: PlayerManager
    + TurnManager(attackQueue: List<Integer>, playerManager: PlayerManager)
    + setNumberOfPlayers(numberOfPlayers: int): void
    + playReverse(): void
    + incrementPlayerTurn(): void
    + incrementAttackCounter(): void
    + setAttackCounter(playerIndex: int): void
    + addAttacks(): void
    + addAttackQueue(attack: int): void
    + removeAttackQueue(): int
    + isAttackQueueEmpty(): boolean
    + setPlayerNumberOfTurns(): void
    + getPlayerTurn(): int
    + setCurrentPlayerNumberOfTurns(numberOfTurns: int): void
    + decrementNumberOfTurns(): void
    + getNumberOfTurns(): int
    + setTurnToTargetedIndexIfAttackOccurred(): void
    + getTurnCountOfPlayer(playerIndex: int): int
    + getAttacked(): boolean
    + getAttackCounter(): int
    + getNumberOfAttacks(): int
    + setNumberOfAttacks(numberOfAttacks: int): void
    + setAttacked(attacked: boolean): void
    + getIsReversed(): boolean
    + startAttackPhase(): void
    + playAttack(): void
    + playTargetedAttack(attackedPlayerIndex: int): void
  }
}

package "ui" {

  class GameController {
    - game: Game
    - gameUI: GameUI
    + GameController(game: Game, gameUI: GameUI)
    + playAction(action: Action): void
    - setupGame(): void
    + run(): void
  }

  class GameSetupUI {
    - game: Game
    - messages: ResourceBundle
    + GameSetupUI(game: Game)
    + chooseLanguage(): ResourceBundle
    + chooseGame(): void
    + chooseNumberOfPlayers(): void
  }

  class GameUI {
    - game: Game
    - messages: ResourceBundle
    - controller: GameController
    + GameUI(game: Game)
    + setController(controller: GameController): void
    + setMessages(messages: ResourceBundle): void
    + startTurn(): void
    + endGame(): void
    + promptForNope(): boolean
    + checkAllPlayersForNope(playerIndex: int): boolean
    - checkValidPlayerIndexInput(): int
    - printPlayerTurn(): void
    - playedCard(): int
    - playNope(playerIndex: int): void
    - playExplodingKitten(int playerIndex): boolean
    - endTurn(): boolean
    - checkIfTheyEndTurn(): boolean
    - playImplodingKitten(card: Card): boolean
    - getLocalizedCardType(cardType: CardType): String
  }

  class Main {
    + {static} main(args: String[]): void
  }

}

package "ui.actions" {

  interface Action {
    {abstract} execute(): void
  }

  class AlterTheFutureAction {
    - game: Game
    - messages: ResourceBundle
    + AlterTheFutureAction(game: Game, messages: ResourceBundle)
    + execute(): void
    - getLocalizedCardType(cardType: CardType): String
  }

  class AttackAction {
    - game: Game
    - messages: ResourceBundle
    - targeted: boolean
    + AttackAction(game: Game, messages: ResourceBundle, targeted: boolean)
    + execute(): void
    - startAttackFollowUp(targeted: boolean): void
    - checkValidPlayerIndexInput(): int
  }

  class CatomicBombAction {
    - game: Game
    - messages: ResourceBundle
    + CatomicBombAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class CurseOfTheCatButtAction {
    - game: Game
    - messages: ResourceBundle
    + CurseOfTheCatButtAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class DrawFromBottomAction {
    - game: Game
    - messages: ResourceBundle
    + DrawFromBottomAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class GarbageCollectionAction {
    - game: Game
    - messages: ResourceBundle
    + GarbageCollectionAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class MarkAction {
    - game: Game
    - messages: ResourceBundle
    + MarkAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class ReverseAction {
    - game: Game
    - messages: ResourceBundle
    + ReverseAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class SeeTheFutureAction {
    - game: Game
    - messages: ResourceBundle
    + SeeTheFutureAction(game: Game, messages: ResourceBundle)
    + execute(): void
    - getLocalizedCardType(cardType: CardType): String
  }
```

```
  class ShuffleAction {
    - game: Game
    - messages: ResourceBundle
    + ShuffleAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class SkipAction {
    - game: Game
    - messages: ResourceBundle
    + SkipAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class SpecialComboAction {
    - game: Game
    - messages: ResourceBundle
    + SpecialComboAction(game: Game, messages: ResourceBundle)
    + execute(): void
    - playTwoCardCombo(): void
    - playThreeCardCombo(): void
    - getCatCardType(comboSize: int): CardType
  }

  class SwapTopAndBottomAction {
    - game: Game
    - messages: ResourceBundle
    + SwapTopAndBottomAction(game: Game, messages: ResourceBundle)
    + execute(): void
  }

  class ThreeCardComboAction {
    - game: Game
    - messages: ResourceBundle
    - cardType: CardType
    + ThreeCardComboAction(game: Game, messages: ResourceBundle, cardType:
CardType)
    + execute(): void
  }

  class TwoCardComboAction {
    - game: Game
    - messages: ResourceBundle
    - cardType: CardType
    + TwoCardComboAction(game: Game, messages: ResourceBundle, cardType:
CardType)
    + execute(): void
  }
}

' Relationships
Game --> Deck
Game --> PlayerManager
Game --> TurnManager
Game --> Instantiator
Game ..> GameRules
```

```
PlayerManager --> Player
PlayerManager ..> Instantiator

TurnManager --> PlayerManager

Player --> Card
Player ..> Instantiator

Deck --> Card
Deck ..> Instantiator

Card --> CardType


GameController --> Game
GameController --> GameUI
GameController ..> Action
GameController ..> GameSetupUI

GameSetupUI --> Game

GameUI --> Game

Main ..> GameController
Main ..> GameUI
Main ..> Game
Main ..> Deck
Main ..> Instantiator

Action <|.. AlterTheFutureAction
Action <|.. AttackAction
Action <|.. CatomicBombAction
Action <|.. CurseOfTheCatButtAction
Action <|.. DrawFromBottomAction
Action <|.. GarbageCollectionAction
Action <|.. MarkAction
Action <|.. ReverseAction
Action <|.. SeeTheFutureAction
Action <|.. ShuffleAction
Action <|.. SkipAction
Action <|.. SpecialComboAction
Action <|.. SwapTopAndBottomAction
Action <|.. ThreeCardComboAction
Action <|.. TwoCardComboAction

SpecialComboAction ..> TwoCardComboAction
SpecialComboAction ..> ThreeCardComboAction

AlterTheFutureAction ..> Game
AttackAction ..> Game
CatomicBombAction ..> Game
CurseOfTheCatButtAction ..> Game
DrawFromBottomAction ..> Game
GarbageCollectionAction ..> Game
MarkAction ..> Game
ReverseAction ..> Game
SeeTheFutureAction ..> Game
```

```
ShuffleAction ..> Game
SkipAction ..> Game
SwapTopAndBottomAction ..> Game
ThreeCardComboAction ..> Game
TwoCardComboAction ..> Game
Action ..> Game

Game --> GameType
Deck --> GameType
Player ..> CardType
GameUI ..>CardType
Instantiator ..> CardType

@enduml
```