

CS396 Software Design Principles and Practices

Project: Milestone 1 Report

People

Evelyn Hung

Option

Option 3

Reference

1. Asked Gemini: "Please break down the activities needed to complete this project and a timeline.
What I ought to do is to let the unfinished parts said in the following be finished in 2 weeks. The steps I could think of that I haven't do: see what design pattern could be used to improve the design, provide an improved design (analysis and diagram), check if it fits design principles, refactor each use case, final implementation and testing, review, report, submit."

Link to Project GitHub Repository

https://github.com/nu-cs-sw-design/project-20252601-396_final.git

Project Planning

Week 1: Design Validation & Core Refactoring

- **Activity 1: Apply and Validate Design Patterns (11/17)**
 - Create "improved design" diagram.
 - Formally identify the patterns used or could be used
 - Write a brief justification for why these patterns are an improvement (to use in final report).
- **Activity 2: Check Design Principles (SOLID) (11/17)**
 - Check the new design against key principles. Ask questions like:
 - **Single Responsibility:** Does my Game class still do too much?
 - **Open/Closed:** If I add a new card, do I have to modify the Game class?
- **Activity 3: Create a Refactoring Plan (11/18)**
 - Based on the design, list the exact classes needed to create, modify, or delete.
 - Prioritize the order. The foundation needs to be built first (e.g., create interface for classes) before refactoring the cards that use it.
- **Activity 4: Refactor Exploding Kitten & Shuffle (11/19, 11/20)**
 - **Exploding Kitten:** Refactor the Game.drawCard() logic. Ensure it correctly checks for DEFUSE, prompts for re-insertion (via GameUI), and kills the player if no DEFUSE is present.

- **Shuffle:** Refactor the card-playing logic to call Game.shuffleDeck() and ensure the Nope flow is triggered.
- **Activity 5: Refactor SwapTopAndBottom (11/21)**
 - Refactor the card-playing logic to call Game.swapTopAndBottom().
 - Ensure the "not enough cards" exception (Exception Flow A) is handled.
- **Activity 6: Refactor Nope Card (11/22, 11/23)**
 - Refactor the GameUI.checkAllPlayersForNope() loop.
 - Ensure the logic for "odd vs. even" nopes is clean, and that the "Nope-ing a Nope" flow works correctly.
 - Make sure Exploding Kitten and Defuse cards cannot be noped.

Week 2: Testing, Documentation & Submission

- **Activity 7: Unit & Integration Testing (11/24-27)**
 - Test each card's functionality in isolation (e.g., "Does playShuffle() actually shuffle the deck?").
 - Play Shuffle -> Nope it -> Check that the deck is not shuffled.
 - Play Swap -> Nope it -> Check that the cards are not swapped.
 - Draw Exploding Kitten with a Defuse -> Check that the player survives and is prompted to re-insert the card.
 - Draw Exploding Kitten without a Defuse -> Check that the player's isDead flag is set.
 - Play Shuffle -> Nope it -> Nope the Nope -> Check that the Shuffle succeeds.
- **Activity 8: Assemble Deliverable 1 (11/28)**
 - Write the 5 use cases (Start Game, Exploding Kitten, Nope, Shuffle, Swap) into a single, clean Software Requirement Document.
- **Activity 9: Assemble Deliverable 2 (11/29)**
 - Create a "Software Design Document."
 - Add "Initial Project" class diagram (a).
 - Add final "Improved" class diagram (b).
 - Write the "Major Changes & Analysis" section (c).
- **Activity 10: Final Review & Submission**
 - Clean up code, add comments, and do a final review.
 - Package SRD, SDD, and the refactored source code into a single submission.
- **Submit.**

Initial Analysis

For Option 3: Refactor a Given Messy Project

The class diagram of the current design:



Use cases:

- **UC1: Start the Game**

Actor: User

Precondition: The application has been launched (`Main.main()` has been executed), and the initial Game, Deck, Player, and GameUI objects have been instantiated.

Basic Flow:

1. System (via GameUI) prompts the user to select a language.
2. User selects a language.
3. System (via GameUI) prompts the user to select a game type (e.g., Exploding Kittens, Streaking Kittens).
4. User selects a game type, which is set in the Game object.
5. System (via GameUI) prompts the user to select the number of players.
6. User selects a valid number of players, which is set in the Game object.
7. System (via Game) gives one DEFUSE card to each active player.
8. System (via Deck) initializes the deck with all cards *except* the Exploding/Imploding Kittens.
9. System (via Game) shuffles the deck.
10. System (via Game) deals the initial hand of cards to each player.
11. System (via Deck) inserts the EXPLODING_KITTEN and other special kitten cards (e.g., IMPLODING_KITTEN) into the deck.
12. System (via Game) shuffles the deck again.
13. System (via GameUI) announces the start of the game and prompts the first player to begin their turn.

Exception Flow A: Invalid Input

- 2.a. If the user provides an invalid input, the system displays an error message.
- 3.a. Resume at Step 1.

Exception Flow B: Invalid Game Type

- 4.a. If the user provides an invalid input, the system displays an error message.
- 5.a. Resume at Step 3.

Exception Flow C: Invalid Number of Players

- 6.a. If the user provides an invalid number of players (e.g., less than 2), the system displays an error message.
- 7.a. Resume at Step 5.

Postcondition: The game is fully initialized. The deck is populated and shuffled, players have their starting hands (including one DEFUSE card), and the game has entered the main turn loop, starting with Player 1.

- **UC2: Draw an Exploding Kitten**

Actor: Current Player

Precondition:

1. It is the Player's turn.
2. The Player has finished playing cards (or chooses not to play any).
3. The Player must draw a card to end their turn (`game.getNumberOfTurns() > 0`).

Basic Flow (Player has a Defuse Card):

1. Player initiates the "draw card" action to end their turn.
2. System (via `Game.drawCard()`) draws the top card from the Deck.
3. System identifies the drawn card is an EXPLODING_KITTEN.
4. System (via GameUI) immediately informs the Player they drew an EXPLODING_KITTEN.
5. System (via `Game.getPlayerAtIndex()`) checks the Player's hand and identifies they have a DEFUSE card.
6. System (via GameUI) informs the Player their DEFUSE card is being used automatically.
7. System (via `Game.removeCardFromHand()`) removes one DEFUSE card from the Player's hand.
8. System (via GameUI) prompts the Player to provide an index (a number) for where to place the EXPLODING_KITTEN back into the Deck.
9. Player provides a valid index.
10. System (via `Game.insertExplodingKittenBack()`) re-inserts the EXPLODING_KITTEN into the Deck at the specified location.
11. System (via GameUI) confirms the card was defused and re-inserted.
12. The Player's turn ends.

Exception Flow A (Player has no Defuse Card):

- 5.a. System (via Game) checks the Player's hand and finds no DEFUSE card.
- 6.a. System (via GameUI) informs the Player they have no DEFUSE card and have "exploded."
- 7.a. System (via `Player.setIsDead(true)`) sets the Player's status to "dead."
- 8.a. The Player's turn ends, and they are out of the game.

Exception Flow B (Invalid Index for Re-insertion):

- 9.a. Player provides an invalid index (e.g., negative, or greater than the deck size).
- 10.a. System (via GameUI) informs the Player the index is invalid.
- 11.a. Resume at Step 8.

Postcondition: Either the Player has been eliminated from the game, or the Player has used one DEFUSE card and returned the EXPLODING_KITTEN to the Deck. The turn passes to the next player.

- **UC3: Play a Nope Card**

Actor: Player (Any player)

Precondition:

1. The current player has just played an action card.
2. The action of that card (the "Original Action") has not yet resolved.
3. The system (via GameUI) is in the "checking for Nopes" phase, prompting all players for a response.

Basic Flow (Playing a Nope to Cancel an Action):

1. System (via GameUI.checkAllPlayersForNope()) iterates through all players (starting from the Action Player), asking if they wish to play a NOPE card.
2. A "Nope Player" (any player with a NOPE card) chooses to play it.
3. System (via Game.playCard()) removes the NOPE card from the Nope Player's hand.
4. System (via GameUI) announces that the Original Action has been Noped.
5. System **restarts the "checking for Nopes" loop** (at Step 1), because this new NOPE card is itself an action that can be Noped by any player.

Alternate Flow A (A Nope is Noped):

1. The Basic Flow (above) occurs: Player A plays an ATTACK, and Player B plays a NOPE.
2. The System restarts the "checking for Nopes" loop (Basic Flow, Step 5).
3. System (via GameUI.checkAllPlayersForNope()) asks all players if they want to NOPE Player B's NOPE.
4. Player A (or any other player, Player C) chooses to play a *second* NOPE.
5. System (via Game.playCard()) removes this second NOPE card from their hand.
6. System (via GameUI) announces that Player B's NOPE has been Noped.
7. System **restarts the "checking for Nopes" loop again** (at Step 1), as this *second* NOPE can also be Noped.

Alternate Flow B (No Nopes are Played / The Loop Finishes):

1. System (via GameUI.checkAllPlayersForNope()) iterates through all players, asking if they wish to play a NOPE.
2. All players decline (or do not have a NOPE card).
3. The "checking for Nopes" loop terminates.
4. System (via GameUI) determines the total number of NOPE cards played in the entire sequence.
5. If the final count of NOPE cards is **odd** (e.g., 1, 3, 5), the Original Action is **canceled**.
6. If the final count of NOPE cards is **even** (e.g., 0, 2, 4), the Original Action **succeeds**, and its effect is immediately triggered.

Exception Flow A (Action Cannot Be Noped):

- 1.a. The "Original Action" is drawing an EXPLODING_KITTEN or playing a DEFUSE card.
- 2.a. The System identifies that these actions cannot be Noped.
- 3.a. The System *skips* this entire use case and proceeds directly to the card's effect (e.g., the "Draw an Exploding Kitten" use case).

Postcondition: The "checking for Nopes" phase is complete. The system has resolved whether the Original Action is canceled or allowed to proceed. All NOPE cards played during this sequence are in the discard pile.

- **UC4: Play a Shuffle Card**

Actor: Current Player

Precondition:

1. It is the Player's turn (game.getPlayerTurn() matches the Player).
2. The Player has one or more SHUFFLE cards in their hand.
3. The Player has not yet drawn a card to end their turn.

Basic Flow (Action Succeeds):

1. Player chooses to play the SHUFFLE card from their hand.
2. System (via GameUI.playedCard()) processes the card selection.
3. System (via Game.playCard()) removes the SHUFFLE card from the Player's hand and places it in the discard pile.
4. System (via GameUI.checkAllPlayersForNope()) asks all other players if they wish to play a NOPE card.
5. No other players play a NOPE card (or an even number of NOPE cards are played, resulting in the action succeeding).
6. System (via Game.shuffleDeck()) triggers the deck shuffling.
7. The Deck.shuffle() method randomizes the order of all cards currently in the deck.
8. System (via GameUI) announces to all players that the deck has been shuffled.
9. The Player's turn continues (they may play more cards or choose to draw).

Alternate Flow A (Action is Noped):

- 4.a. System (via GameUI.checkAllPlayersForNope()) asks all other players if they wish to play a NOPE card.
- 5.a. Another player plays a NOPE card.
- 6.a. An odd number of NOPE cards are played in total, and the SHUFFLE action is canceled.
- 7.a. The SHUFFLE card remains in the discard pile, but the Game.shuffleDeck() method is **not** called.
- 8.a. System (via GameUI) announces that the SHUFFLE was Noped.
- 9.a. The Player's turn continues.

Postcondition: The SHUFFLE card is in the discard pile. If the action was successful, the deck is now in a new, random order. If the action was Noped, the deck's order is unchanged. The Player's turn continues.

- **UC5: Play a Swap Top And Bottom Card**

Actor: Current Player

Precondition:

1. It is the Player's turn (game.getPlayerTurn() matches the Player).
2. The Player has one or more SWAP_TOP_AND_BOTTOM cards in their hand.
3. The Player has not yet drawn a card to end their turn.

Basic Flow (Action Succeeds):

1. Player chooses to play the SWAP_TOP_AND_BOTTOM card from their hand.
2. System (via GameUI.playedCard()) processes the card selection.
3. System (via Game.playCard()) removes the SWAP_TOP_AND_BOTTOM card from the Player's hand and places it in the discard pile.
4. System (via GameUI.checkAllPlayersForNope()) asks all other players if they wish to play a NOPE card.
5. No other players play a NOPE card (or an even number of NOPE cards are played).
6. System (via Game.swapTopAndBottom()) executes the card's effect.
7. The Game object (via the Deck) draws the top card and the bottom card, then re-inserts the original top card at the bottom and the original bottom card at the top.
8. System (via GameUI) announces to all players that the top and bottom cards of the deck have been swapped.
9. The Player's turn continues (they may play more cards or choose to draw).

Alternate Flow A (Action is Noped):

- 4.a. System (via GameUI.checkAllPlayersForNope()) asks all other players if they wish to play a NOPE card.
- 5.a. Another player plays a NOPE card.
- 6.a. An odd number of NOPE cards are played in total, and the SWAP_TOP_AND_BOTTOM action is canceled.
- 7.a. The Game.swapTopAndBottom() method is **not** called.
- 8.a. System (via GameUI) announces that the action was Noped.
- 9.a. The Player's turn continues.

Exception Flow A (Not Enough Cards in Deck):

- 6.a. The System (via Game.swapTopAndBottom()) checks the deck size and finds it is one or less (checkDeckHasOneCardOrLess()).
- 7.a. The action fails and throws an exception (DECK_HAS_ONE_CARD_EXCEPTION).
- 8.a. System (via GameUI) informs the player the action failed as there are not enough cards in the deck to swap.
- 9.a. The Player's turn continues.

Postcondition: The SWAP_TOP_AND_BOTTOM card is in the discard pile. If the action was successful and valid, the top and bottom cards of the Deck have been exchanged. The Player's turn continues.