Strategic War

An Object-Oriented Software Design Project

By Jared Peterson and Ryan Harper

Table of Contents

Abstract

Introduction

Our group has decided to design a game for desktop/laptop computers. The game we have selected to design is called *Strategic War,* a variant of the traditional *War* card game. *Strategic War* differs from *War* in that each player has a hand of cards to select from instead of always playing the top card on the pile. This allows the game to have an element of strategy instead of being completely random. The game is identical in almost all other aspects.

Goal

The application will contain simple interfaces that will allow the player to start a game, play a game, and quit a game. Each game will be played until either an end condition is met or the player chooses to quit. Since this game requires two players, a simple AI will be developed to play against the player.

Purpose

This application is intended for users of all ages, and is specifically targeted to fans of card games or strategy games. The application will require a user to have the Java Runtime Environment installed.

Execution

The game was implemented using the JDK1.8 and the JavaFX library. The user interface was crafted utilizing the Scene Builder tool and the resulting FXML files were then loaded into JavaFX scenes. The core mechanics of the game were built independent of the graphical components, the only exception being the WarGame class' observer interface. With development time as the primary consideration, complex animations were omitted.

Conclusion

We have learned a lot through the development of this project. One lesson that stands out is the power of using design patterns when designing software. For instance, the use of the singleton pattern for the WarGame class significantly reduced the complexity of passing its reference to multiple dependent components. This lesson was especially powerful because this pattern was only implemented half way through development of the project. After the heartache

of refactoring code was complete, we recognized the simplicity it brought to a once complex system.

Another interesting lesson we learned is just how much design patterns assist in understanding and documenting a codebase. Before using design patterns, our codebase started to slowly become monolithic and new changes only made it more complex. Design patterns broke that down into manageable chunks, even making the class diagram easier to understand.

Finally, we learned the importance of good developer tools. Gluon's Scene Builder was *literally* a game changer. The freedom and flexibility it provided when designing the various menus and scenes allowed us to quickly build visually appealing scenes. This saves us numerous hours, and allowed us to focus on functionality and polish the game in the last stretch of development.

Functional Specification

When the user launches the application from their desktop, it shows a main menu. The 31user is given two options; start a game and quit application. When the user selects start game, they are then shown a difficulty selection menu. Should the user select quit application the application will close. After selecting a difficulty the user confirms the selection and the scene changes to show the game environment.

The application shows visual cues indicating the randomization of the playing deck, the splitting of the playing deck, and the drawing of the player's five card hand. The player selects a card to play while the AI simultaneously chooses its card. Once the player chooses a card the AIs selection is revealed the results of the battle are displayed. A visual indicator is updated to show the player's current score as well as the AI's score. Should the selections result in a war, then two cards are selected at random from both the player's and the AI's hand and are placed face down as prizes. Then both the player and AI select a card from the remaining two cards. The resulting battle is then displayed much like the standard battle however, upon completion the face down cards are revealed. The visual score indicators are updated based on the results.

Once either the player or the AI has gathered all the cards of the deck a victory or defeat menu is shown. Each menu will display the final results as well as how many turns were taken. It will also give options to return to the main menu or to play a rematch. When return to menu is selected the application transitions back to the main menu. When rematch is selected the game restarts with the same AI settings.

At any point the player may quit the current game. Quitting the game transitions the application back to the main menu.

Use Cases

1. Open application

a. User starts application from desktop
b. Application displays game title
c. Application displays main menu

2. Start a game

a. User **opens the application**
b. Press a “Start New Game” button to start a new game of Strategic War
c. User selects difficulty from given options
d. User confirms selection
e. Application changes scenes to game scene

3. Set Up a Game

a. User **starts a game** or **rematch**
b. The user’s hand of 5 cards is shown to the user on the bottom of the screen
c. The AI’s hand of 5 cards is shown face down at the top of the screen
d. The user’s deck is to the right of the user’s hand and shows the card count
e. The AI’s deck is to the left of the AI’s hand and shows the card count
f. The discard piles are to the left of the User’s deck and to the right of the AI’s deck.

4. Fight a battle

a. Application **sets up a game** or game loop continues
b. The user selects a card to play
c. The card is removed from the user’s hand
d. The card appears in the middle, to the right of the opponent’s face up played card
e. The value of the selected cards is compared.

5. User wins a battle

a. User **fights a battle**
b. User’s card value is higher than AI’s card value
c. Applications displays the victory to the user
d. Both the user’s card and AI’s card go into the user’s discard pile

6. User loses a battle

a. User **fights a battle**
b. User’s card value is lower that AI’s card value
c. Application displays the defeat to the user
d. Both the user’s card and AI’s card go into the AI’s discard pile

7. Fight a war

a. User **fights a battle**
b. User's card value is the same as the AI's card value
c. Two cards from both the User's and AI's hand are selected at random
d. The four cards are shown face down next to the battling cards
e. The user and the AI select one of two remaining cards to battle
f. The value of the selected cards is compared

8. User wins a war

a. User **fights a war**
b. User's card value is higher than AI's card value
c. Applications displays the victory to the user
d. Face down cards on each side is revealed
e. All cards go into the user's discard pile

9. User loses a war

a. User **fights a war**
b. User's card value is lower than AI's card value
c. Applications displays the defeat to the user
d. Face down cards on each side is revealed
e. All cards go into the AI's discard pile

10. Fill hand

a. User either **wins a battle**, **loses a battle**, **wins a war**, or **loses a war**
b. The application checks how many cards the User's and AI's hands need
c. The application deals as many cards as needed to make a hand of 5 cards to the AI and to the User at the same time.
d. If the deck is empty before the User's or AI's hand is filled, the discard pile of either the User or AI is shuffled and creates a new deck.
e. If the deck is empty still, the User or AI plays with the remaining hand.

11. User wins a game

a. User either **wins a battle** or **wins a war**
b. The number of cards in the AI's hand, deck, and discard is zero.
c. The application displays victory screen with options for rematch or quit to menu

12. User loses a game

a. User either **loses a battle** or **loses a war**
b. The number of cards in the user's hand, deck, and discard is zero.
c. The application displays defeat screen with options for rematch or quit to menu

13. Rematch

a. User either **wins a game** or **loses a game**
b. Player selects the rematch option

   c. Application **sets up a game** of the same difficulty

14. Quit match
    a. User selects quit button on screen
    b. User confirms action
    c. Application changes scenes to main menu
15. Quit application
    a. User selects quit application button on main menu
    b. User confirms selection
    c. Application gracefully shuts down

Design Specification

CRC Cards

Card	
• Stores a card value (2-10, J, Q, K, A) • Stores a card suit (Hearts, Diamond, Clubs, Spades) • To be compared to other cards by value	• Deck • Hand • DiscardPile • WarGame • Battle • War

ImagePool	
• Instantiate and store a pool of images to be used by the UI	

Deck	
• Stores Cards • Shuffles Cards • Splits deck into separate half decks	• Card

DeckFactory	
• Create a new deck with 52 cards	• Deck • Card

Hand	
• Stores playable Cards	• Card

DiscardPile	
• Stores discarded Cards	• Card

Battle	
• Stores the cards involved • Calculates the winner	• Card

War	
• Stores cards involved in war • Calculates winner	• Card

MainGUI	
• Stores UI elements of the Main scene • Displays UI elements of the Main scene	• ApplicationController

GameGUIController	
• Stores UI elements of the Game scene • Displays UI elements of the Game scene	• WarGame • Card

WarAI	
• Interface for implementing game AI	• WarGame • Card • Hand

EasyAI	
• Takes actions against the player • Uses the lowest Card in the Hand	• WarGame • Card • Hand • WarAI

HardAI	
• Takes actions against the player • Uses the highest Card in the Hand	• WarGame • Card • Hand • WarAI

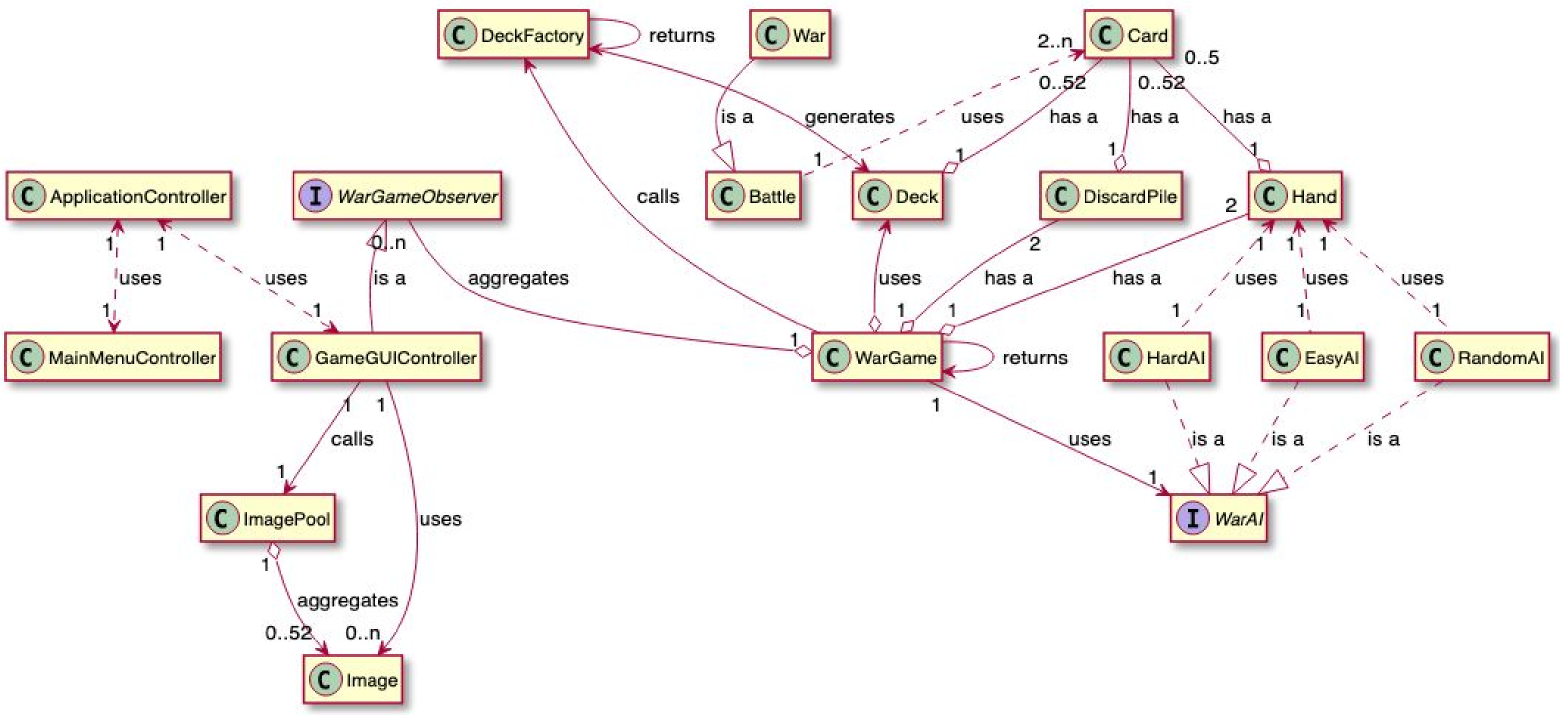
RandomAI	
• Takes actions against the player • Uses a random Card in the Hand	• WarGame • Card • Hand • WarAI

WarGame	
• Stores all game state objects • Controls the main game loop	• WarGameObserver • WarAI • War • Battle • Hand • Deck • DiscardPile

WarGameObserver	
• Provide an interface for GUI components to receive updates on the game	• GameGUIController • WarGame

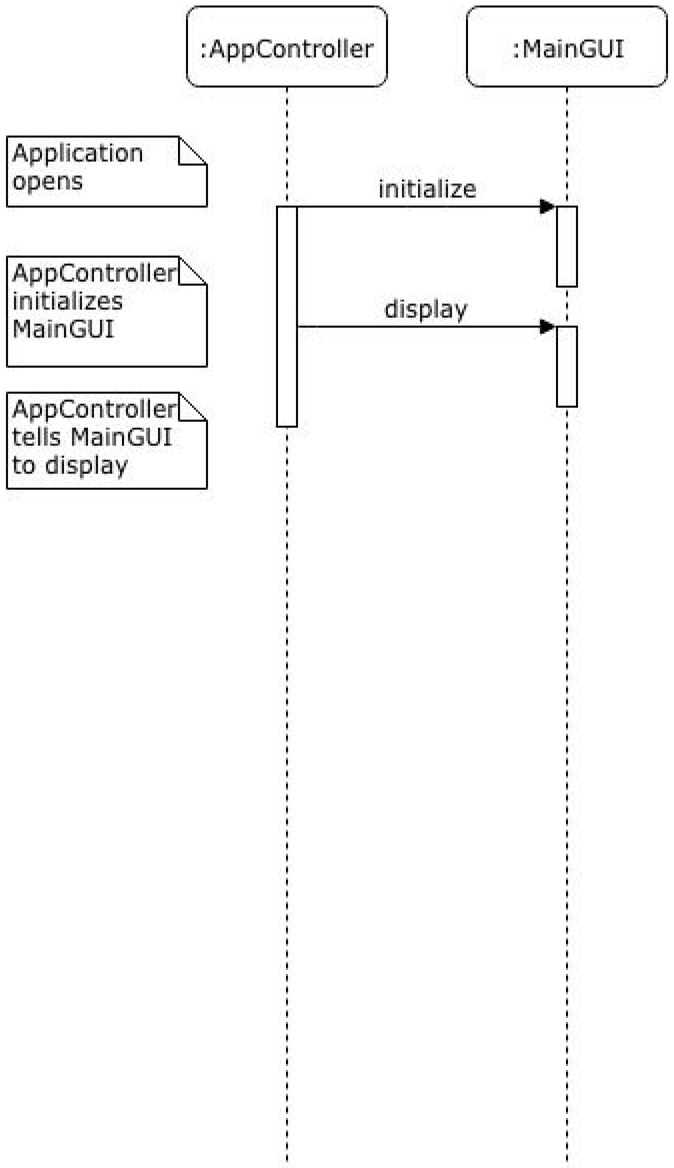
ApplicationController	
• Control major app functions • Transition between UI scenes	• MainMenuController • GameGUIController

Class Diagram

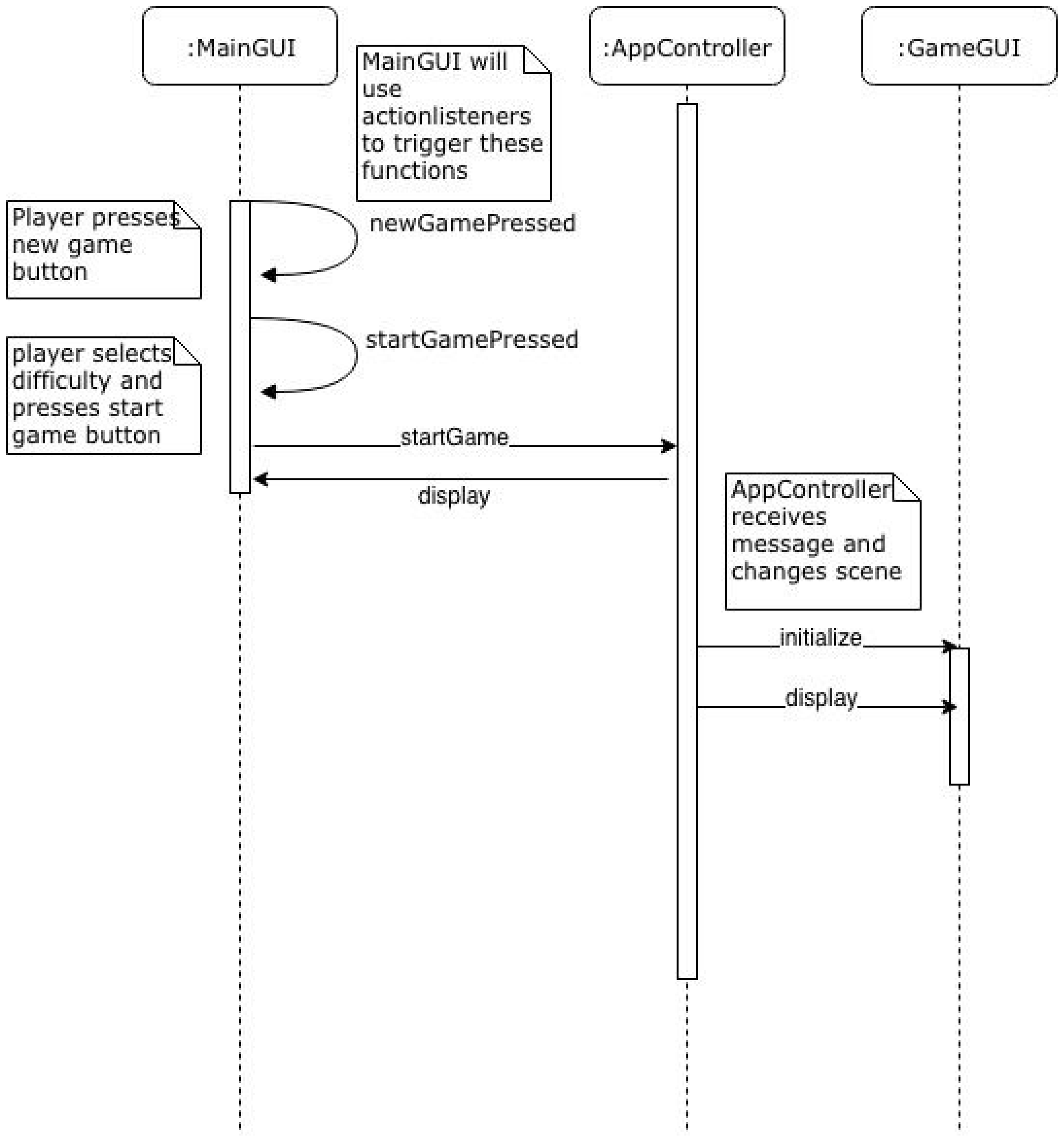


Sequence Diagrams

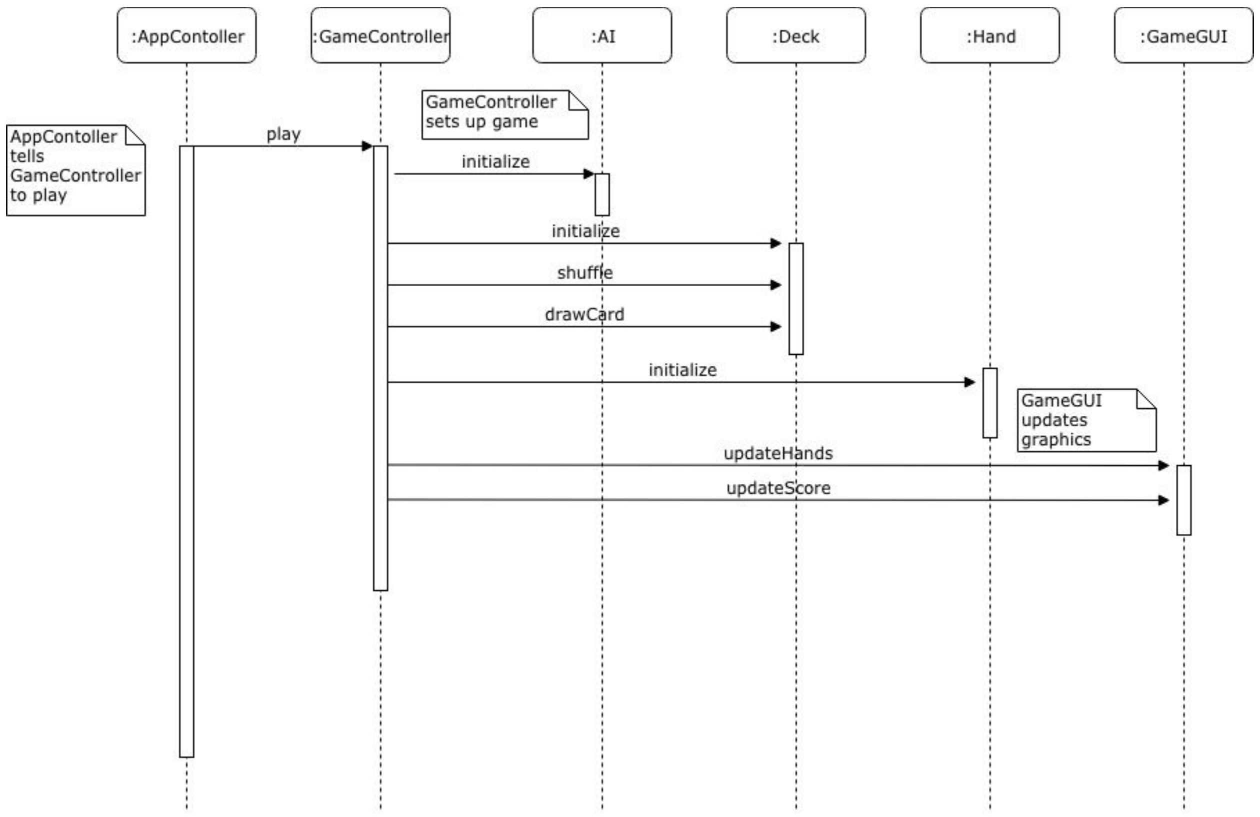
1. Open application


:AppController
:MainGUI
Application opens
initialize
AppController initializes MainGUI
display
AppController tells MainGUI to display

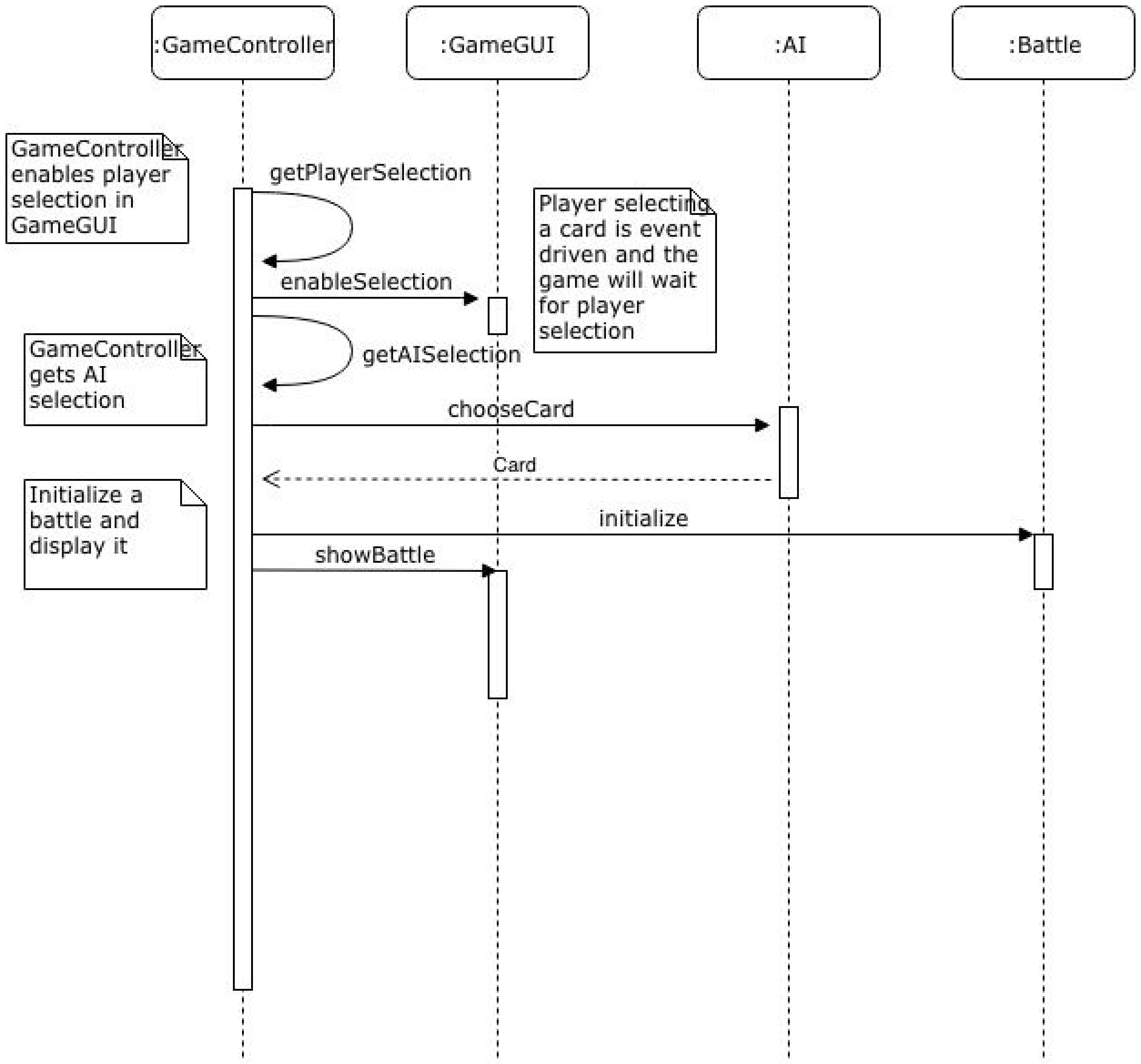
2. Start a Game


:MainGUI
:AppController
:GameGUI
MainGUI will use actionlisteners to trigger these functions
Player presses new game button
newGamePressed
player selects difficulty and presses start game button
startGamePressed
startGame
display
AppController receives message and changes scene
initialize
display

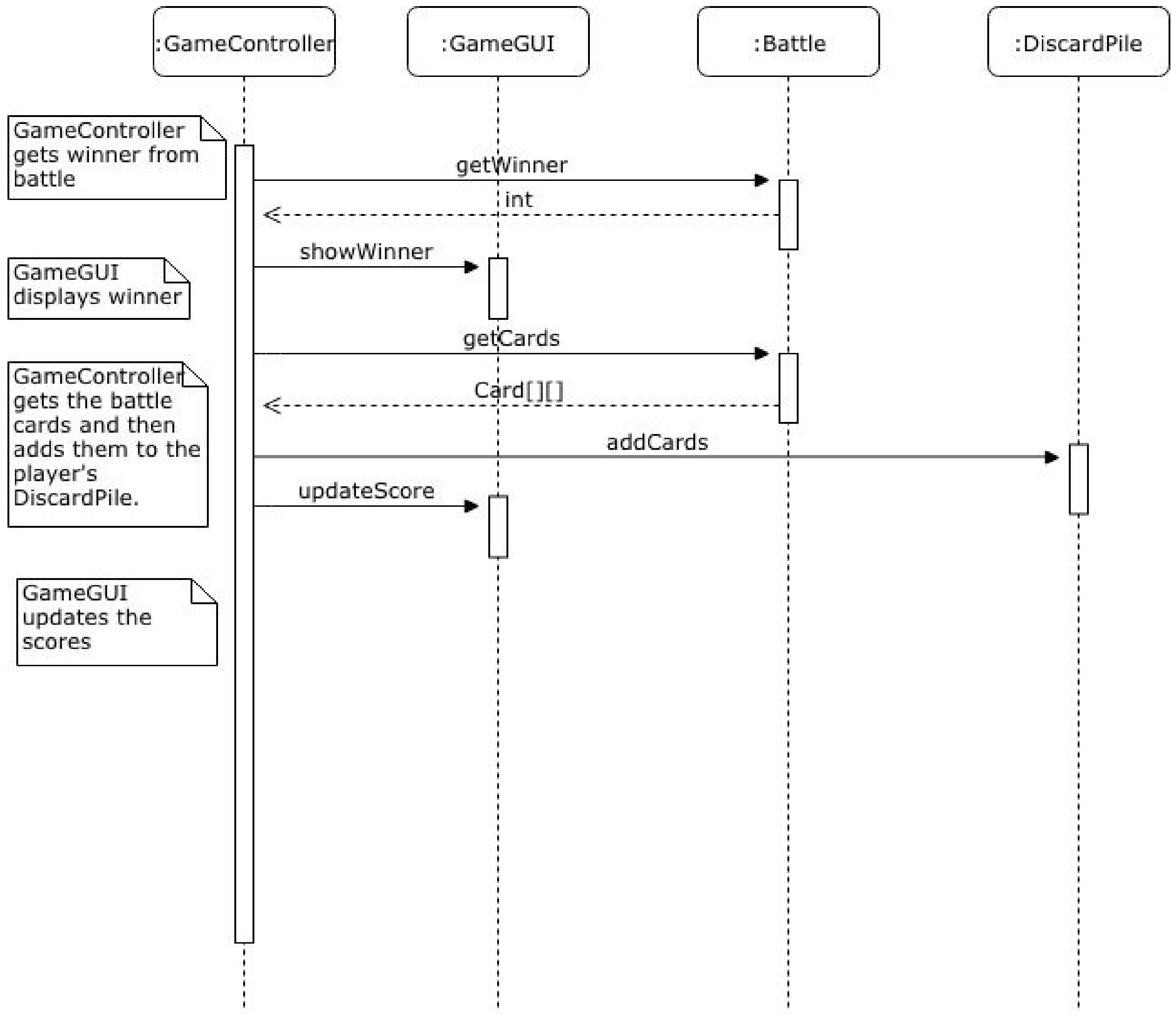
3. Set Up a Game



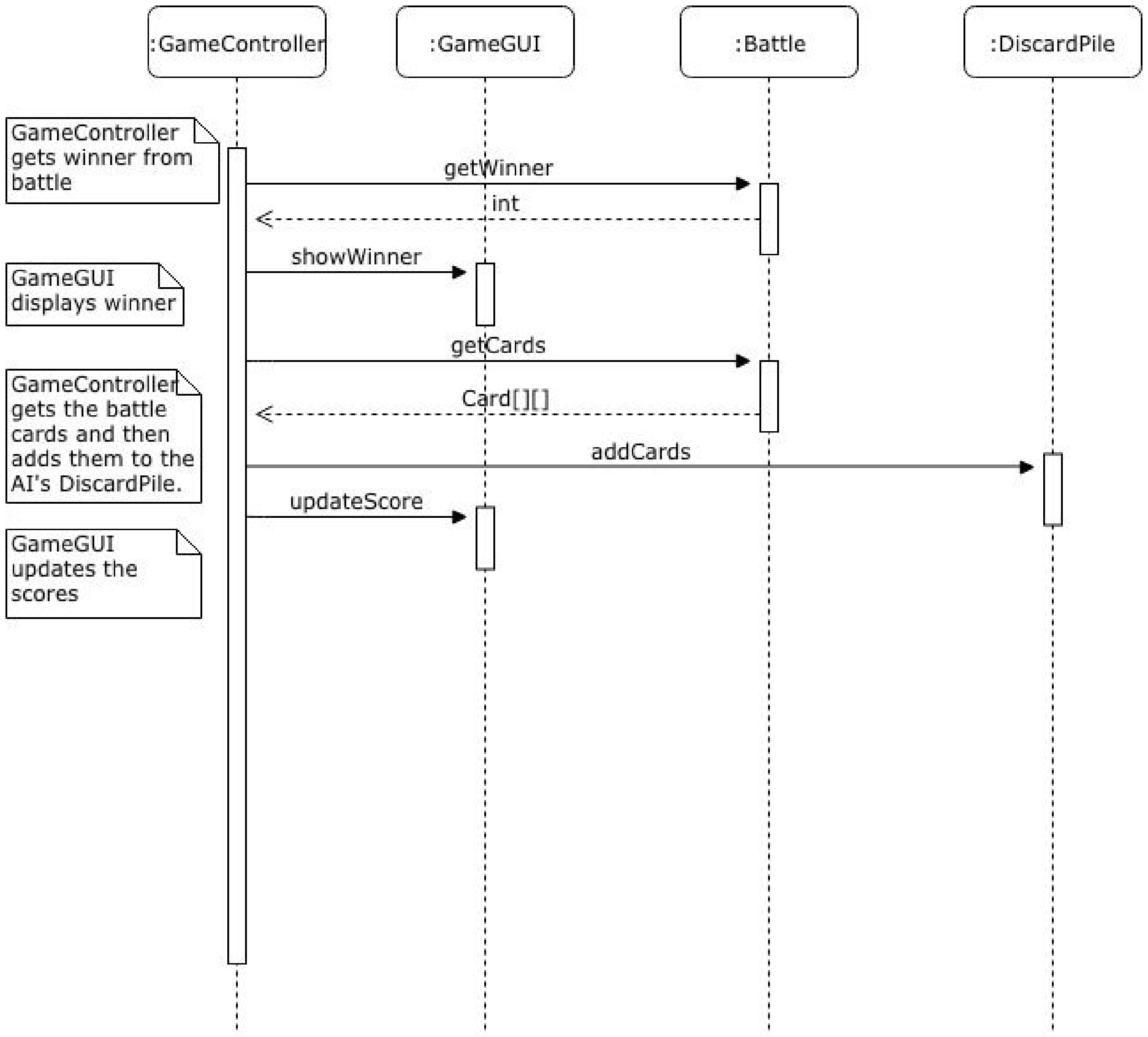
4. Fight a Battle


:GameController
:GameGUI
:AI
:Battle
GameController enables player selection in GameGUI
getPlayerSelection
enableSelection
Player selecting a card is event driven and the game will wait for player selection
GameController gets AI selection
getAISelection
chooseCard
Card
initialize
Initialize a battle and display it
showBattle

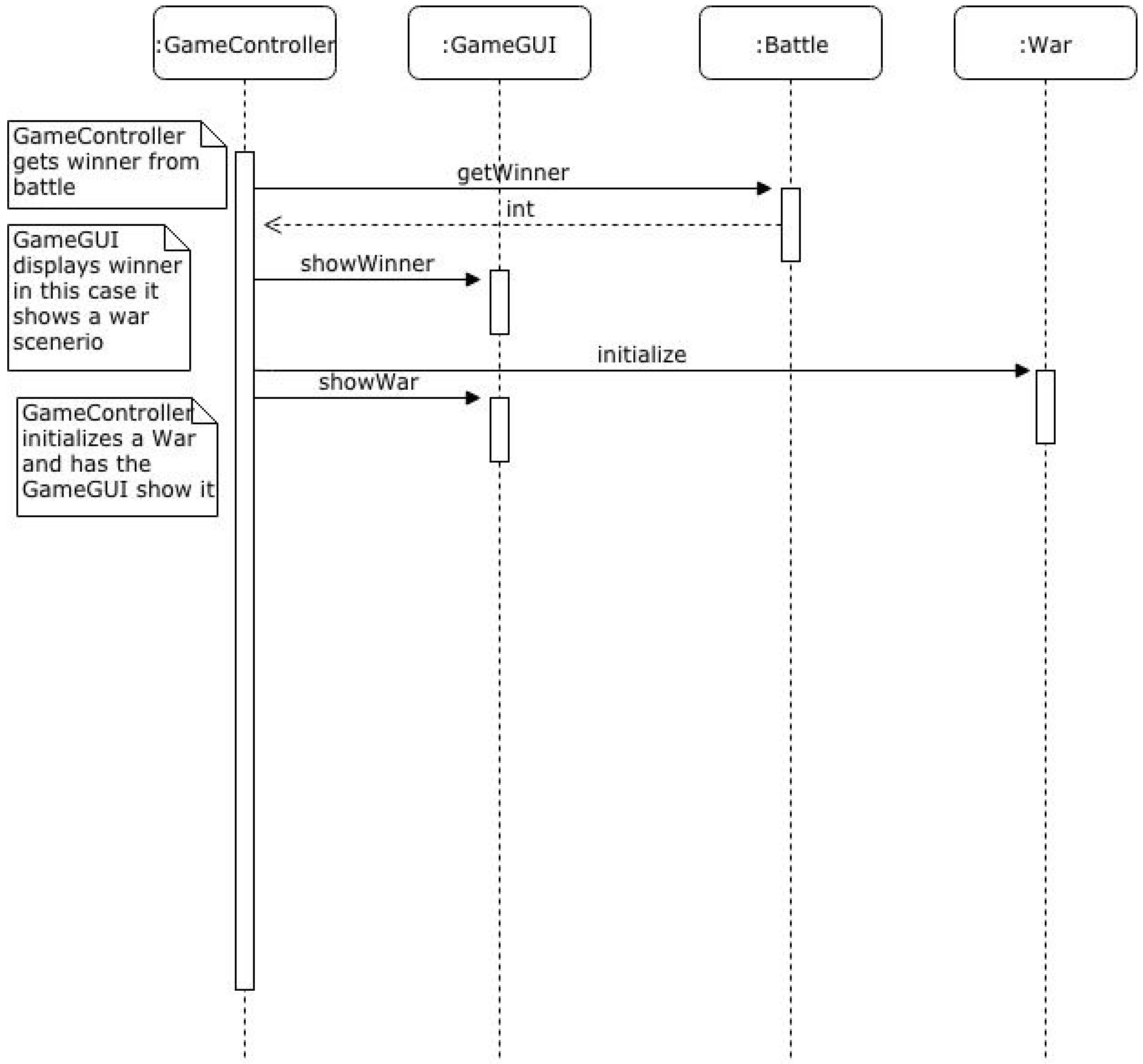
5. User Wins a Battle


:GameController
:GameGUI
:Battle
:DiscardPile
GameController gets winner from battle
getWinner
int
showWinner
GameGUI displays winner
getCards
Card[][]
GameController gets the battle cards and then adds them to the player's DiscardPile.
addCards
updateScore
GameGUI updates the scores

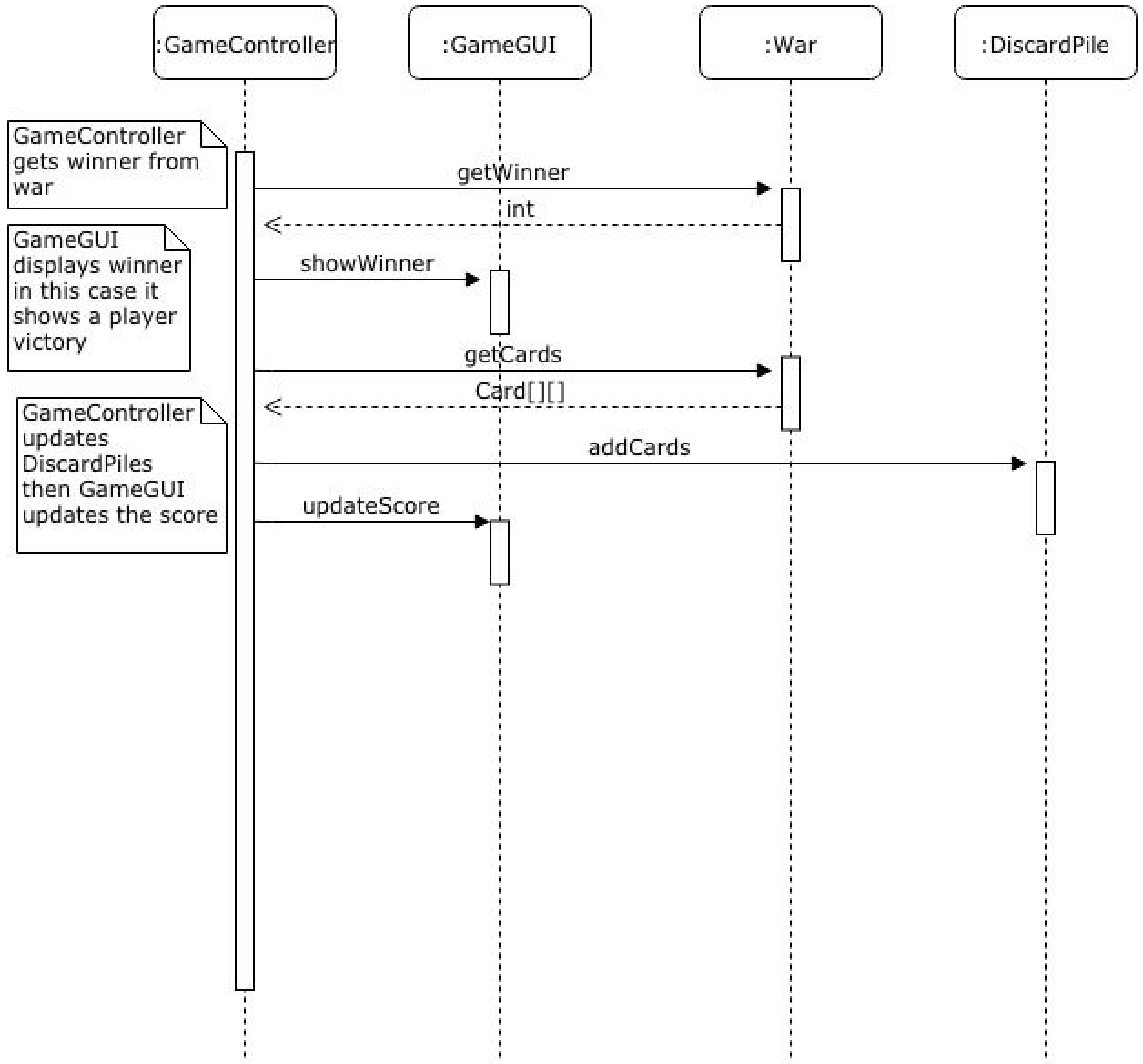
6. User Loses a Battle


:GameController
:GameGUI
:Battle
:DiscardPile
GameController gets winner from battle
getWinner
int
showWinner
GameGUI displays winner
getCards
Card[][]
GameController gets the battle cards and then adds them to the AI's DiscardPile.
addCards
updateScore
GameGUI updates the scores

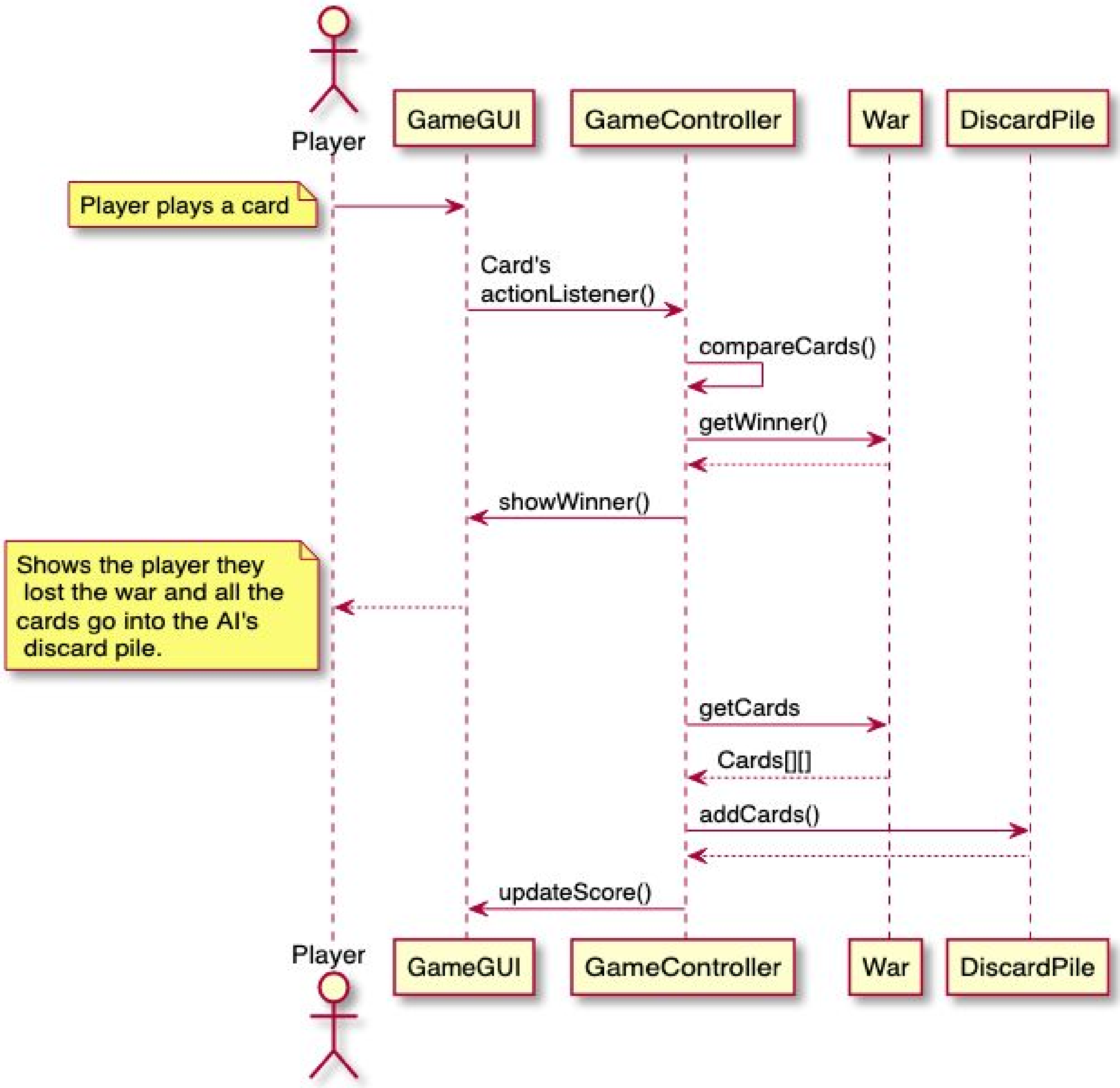
7. Fight a War


:GameController
:GameGUI
:Battle
:War
GameController gets winner from battle
GameGUI displays winner in this case it shows a war scenerio
GameController initializes a War and has the GameGUI show it
getWinner
int
showWinner
initialize
showWar

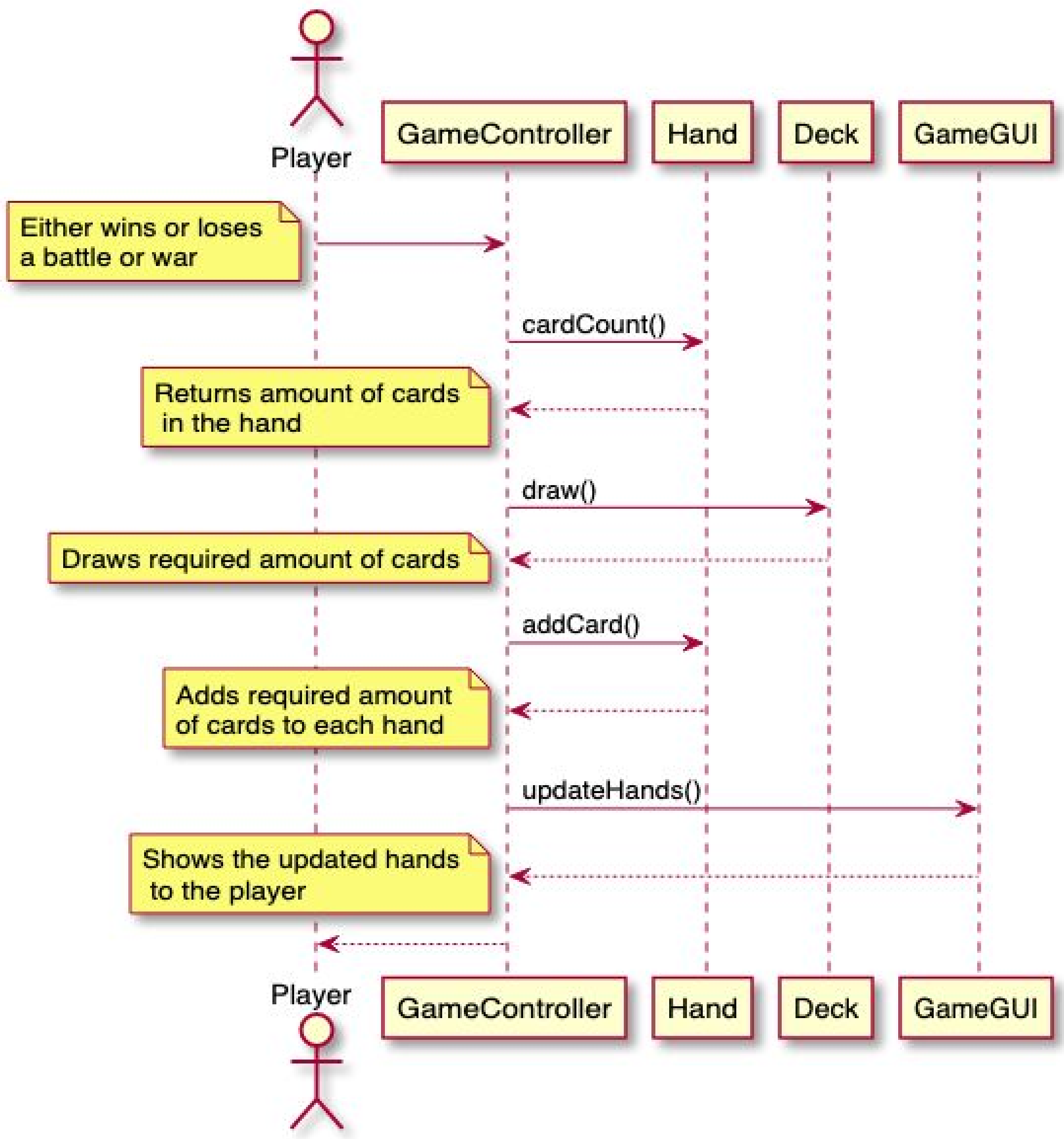
8. User Wins a War


:GameController
:GameGUI
:War
:DiscardPile
GameController gets winner from war
GameGUI displays winner in this case it shows a player victory
GameController updates DiscardPiles then GameGUI updates the score
getWinner
int
showWinner
getCards
Card[][]
addCards
updateScore

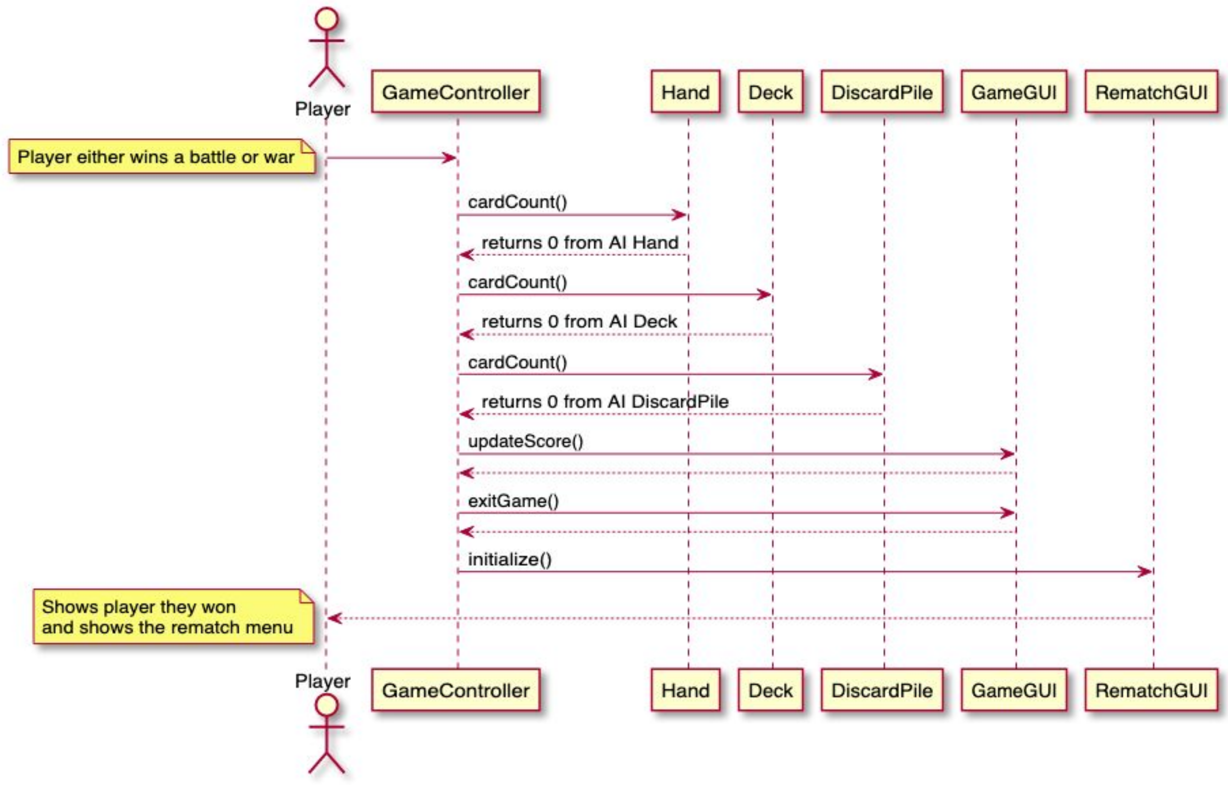
9. User Loses a War



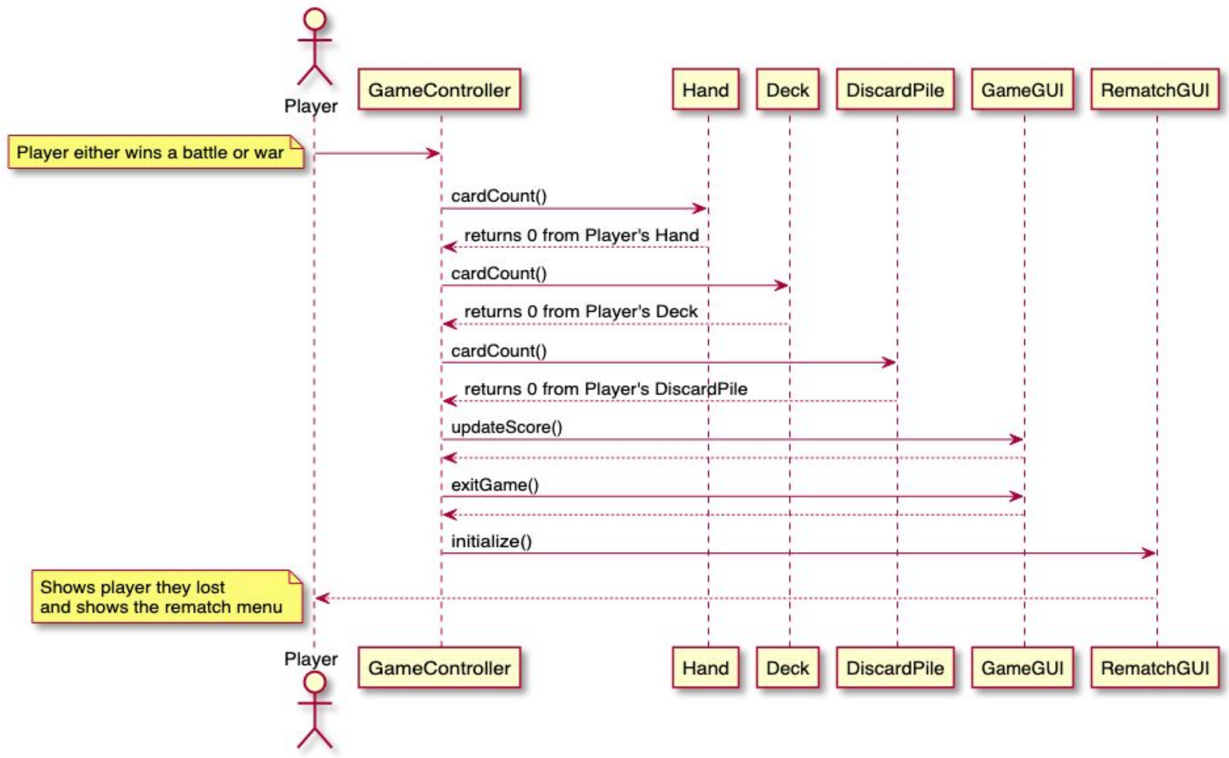
10. Fill Hand


Player
GameController
Hand
Deck
GameGUI
Either wins or loses a battle or war
cardCount()
Returns amount of cards in the hand
draw()
Draws required amount of cards
addCard()
Adds required amount of cards to each hand
updateHands()
Shows the updated hands to the player
Player
GameController
Hand
Deck
GameGUI

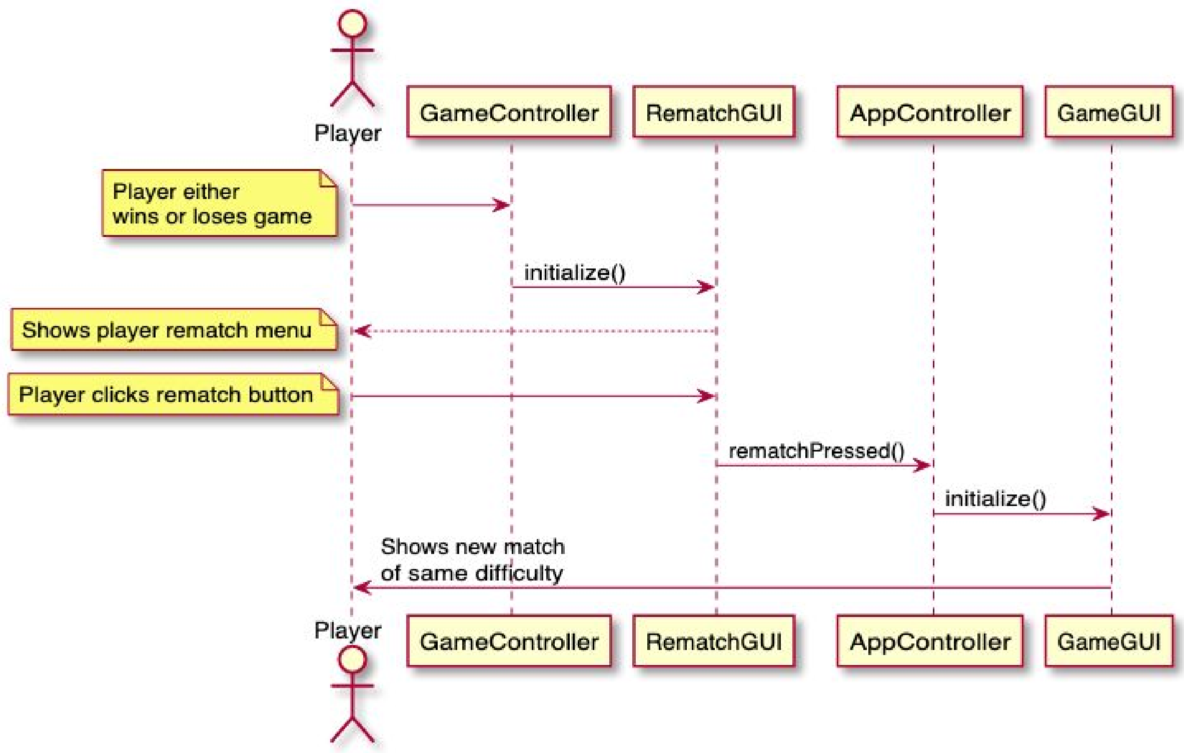
11. User Wins a Game



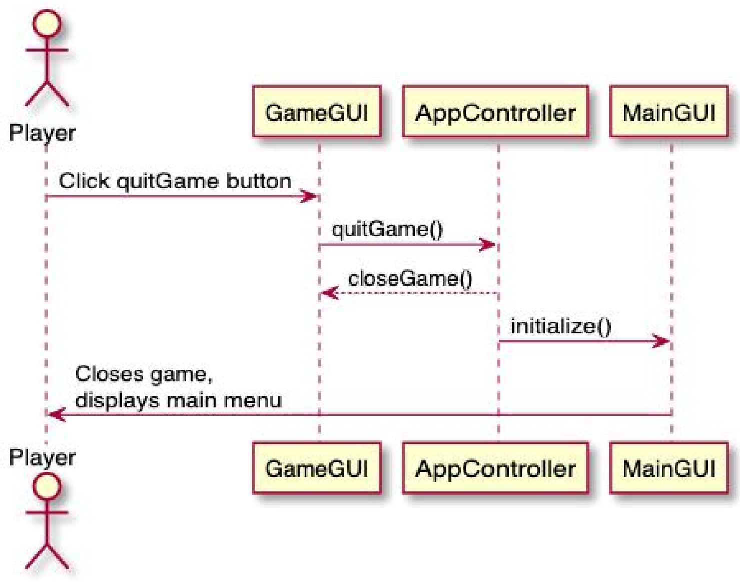
12. User Loses a Game


Player
GameController
Hand
Deck
DiscardPile
GameGUI
RematchGUI
Player either wins a battle or war
cardCount()
returns 0 from Player's Hand
cardCount()
returns 0 from Player's Deck
cardCount()
returns 0 from Player's DiscardPile
updateScore()
exitGame()
initialize()
Shows player they lost and shows the rematch menu

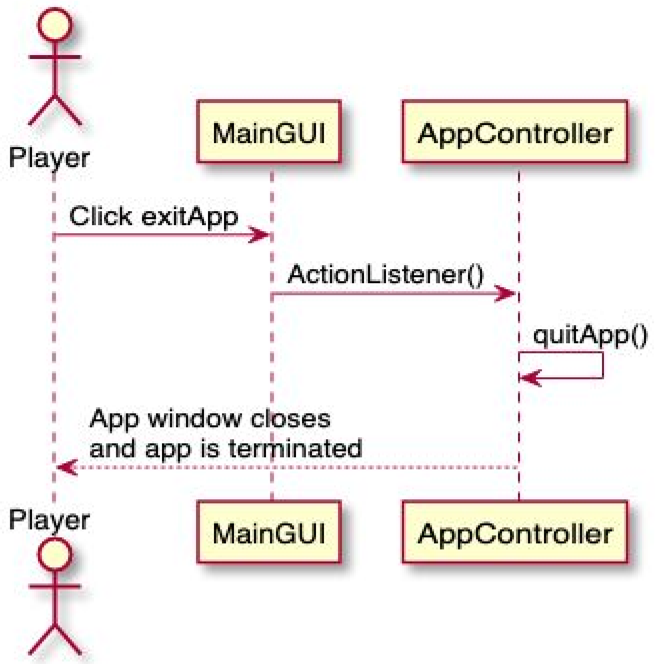
13. Rematch



14. Quit Match

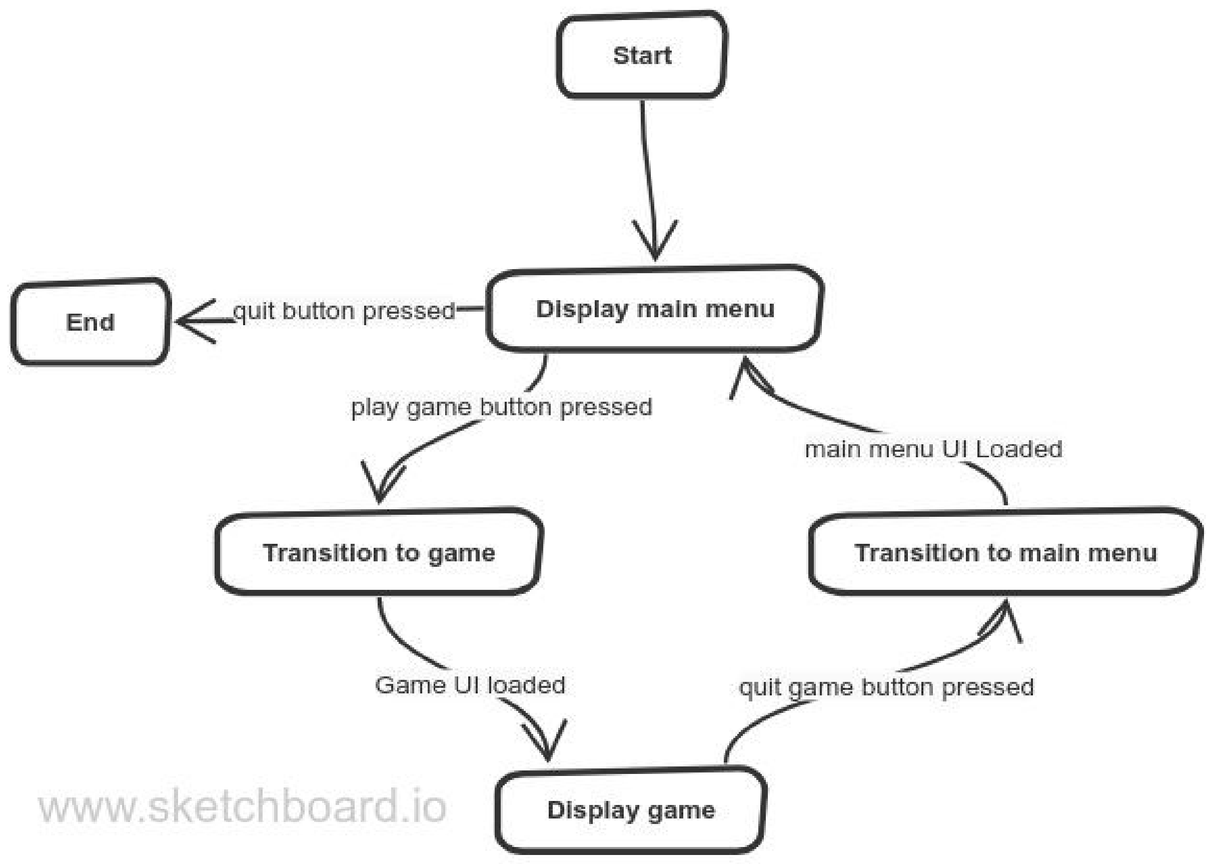


15. Quit Application

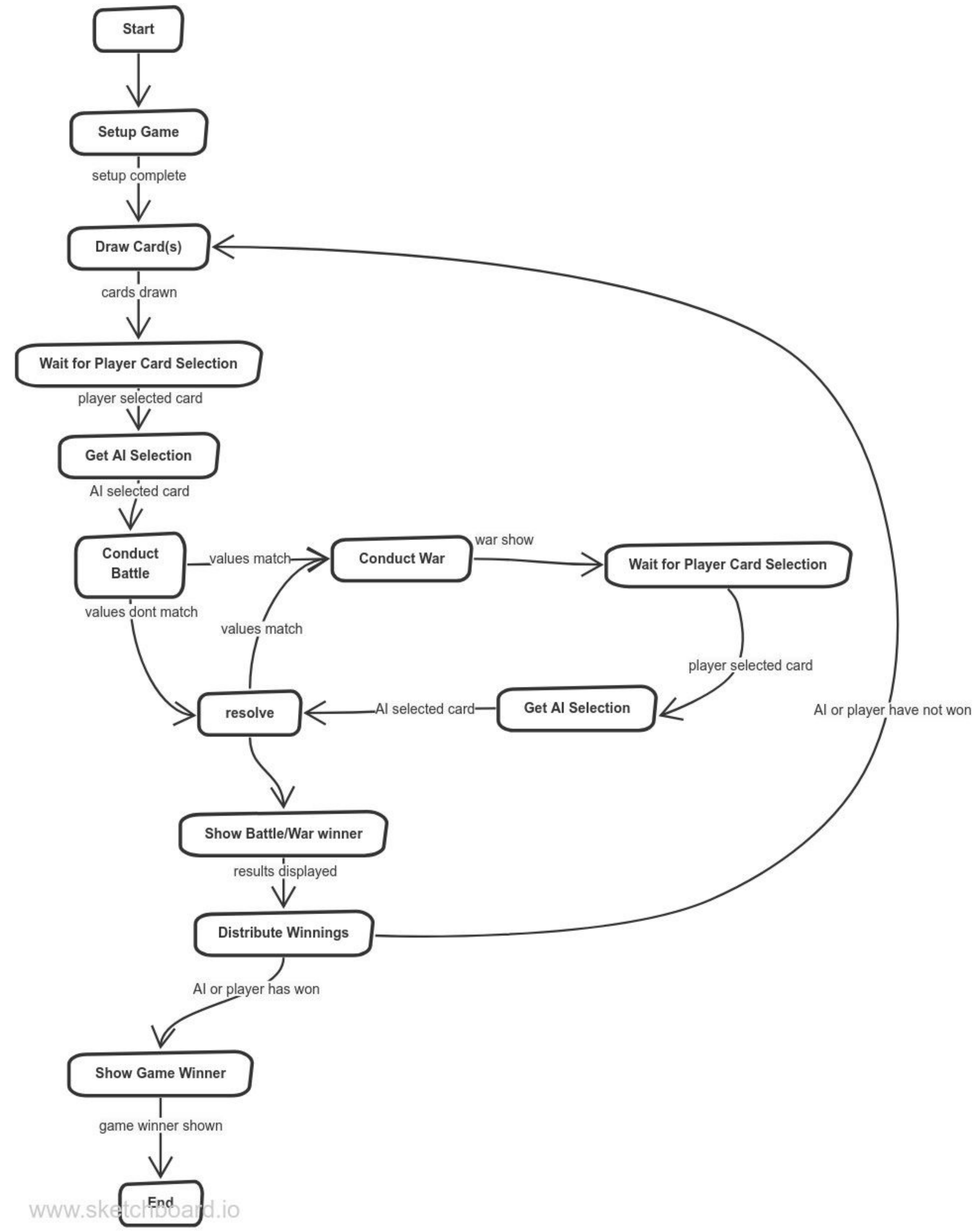


State Diagrams

AppController



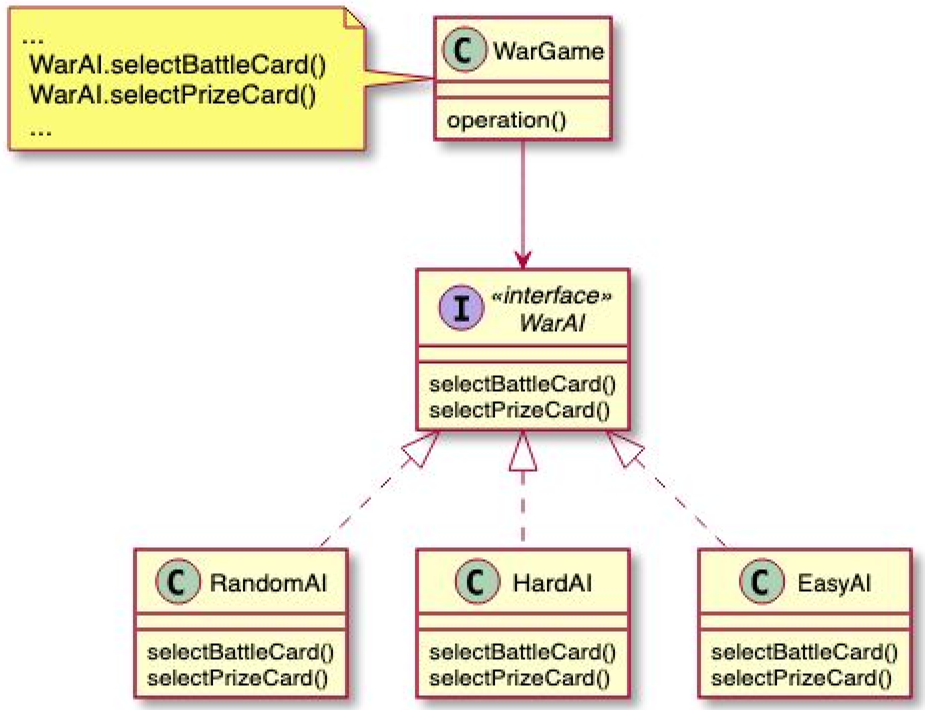
GameController



Design Patterns

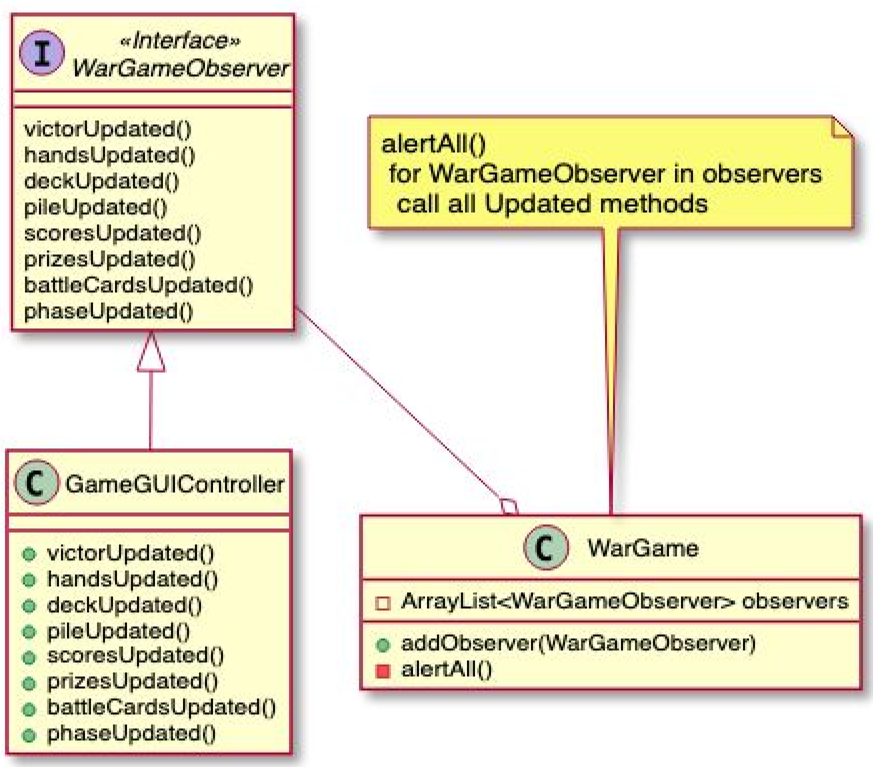
Strategy Pattern

In our project, we implemented the Strategy pattern where the Context is our WarAI class, the Strategy is our WarAI class and the Concrete Strategies are the RandomAI, HardAI, and EasyAI classes. The WarGame class benefits from having different variants of the selectBattleCards() and selectPrizeCards() algorithms. Also, if someone wanted to implement their own WarAI implementation, they could make a completely custom AI.



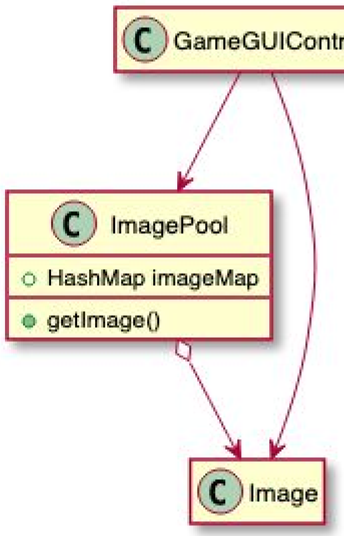
Observer Pattern

We also implemented the Observer pattern where the Subject is WarGame, the Observer is WarGameObserver, and the Concrete Observer is GameGUIController. WarGame is the source of events and the GameGUIController wants to know about these events.



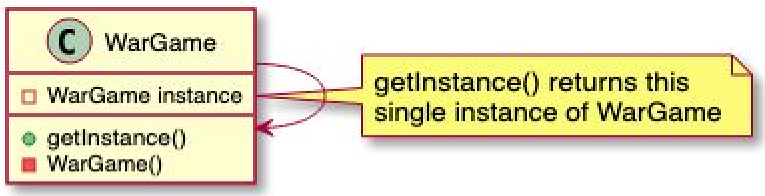
Object Pool Pattern

We implement the Object Pool pattern where the Client is GameGUIController, the Object Pool is ImagePool, and the Reusable Pool is the JavaFX class Image. We did this because we wanted to initialize a finite amount of Images all at once in the beginning and pull from the pool of images rather than generate them on the fly.



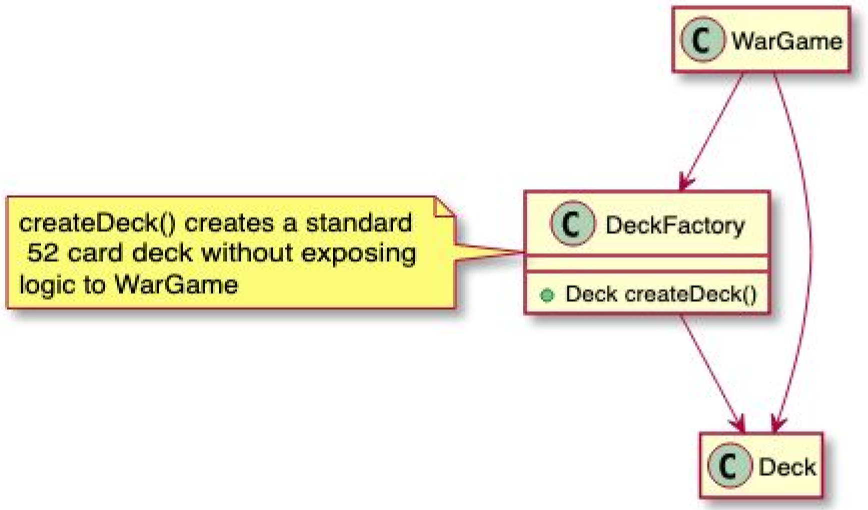
Singleton Pattern

We have a couple examples of Singleton, but we'll just focus on WarGame. With WarGame, we want only one instance running at any time and don't want to create any more. This class stores a single instance of itself privately and either creates it and stores it if it doesn't exist yet or returns it when getInstance() is called.



Factory Pattern

With the Factory pattern, the intent is to create objects without exposing the logic behind the instantiation. With that in mind, WarGame (the Client) calls upon the DeckFactory (the Factory) to create a Deck (the Product) to use. DeckFactory creates a standard 52 card deck without exposing that underlying logic to WarGame.


WarGame
DeckFactory
Deck createDeck()
Deck
createDeck() creates a standard 52 card deck without exposing logic to WarGame