3. WRITTEN RESPONSES

3 a.

3.a.i.

My app is a strategy-based card game, “4 Colors,” where the objective is to place your entire deck before all other opponents prior to the clock hitting zero. All players initially have seven cards in their deck. If all cards are discarded before everybody else, you win, otherwise it is a loss. If no players obtain victory and the clock reaches zero, it is a draw. The game cards consist of four colors: red, blue, green and yellow, along with numbers ranging from one to nine. Special power cards are also scattered throughout the decks, these include: reverse, +4, +2, and WILD. Only cards of the same attributes can be played on top of each other, unique gameplay rules are described in the application. You can cycle your deck with the arrows. The stack in the middle shows the active card and a personal card count is above your deck.

3.a.ii.

The video demonstrates most of the functionalities of my application, the primary one being the card game itself, although along with this I preview a settings screen, credits, the main menu, and a how-to guide. Gameplay is shown throughout the video. Demonstrated in the gameplay is the countdown clock, cards being controlled through drag events, player turns, a deck counter, card cycling, and other algorithmically controlled game rules. Additional options are placed in the settings menu to allow for some user customization. The guide page is provided to explain advanced game rules and controls. The main menu serves as a hub for the application, listing all the individual screens. Most screens contain a back button, so the user can navigate backwards. The credits screen lists of all rights and contributions. Not shown in the video due to the length restraint is winning and losing screens, which the game ends with.

3.a.iii.

The user's button presses and mouse movement serve as the input, however the output varies depending on the chosen event(s). For the primary aspect of my application it is gameplay, the user clicks a card in their own deck on their turn and drags it to the center, then clicks it to place. The user is able to select the card type of choice from their deck, varying input. The output is found algorithmically, in this case it is usually the cycling turns of the other three users' taking place and the effect it’ll have on the player. They may possibly have to draw cards, or choose another card out of their deck. Throughout this lengthy series of inputs and outputs is determined a win or loss, displayed on an end screen. Less complex instances such as setting toggles and navigation exist too, which output in property change(s) and screen(s).

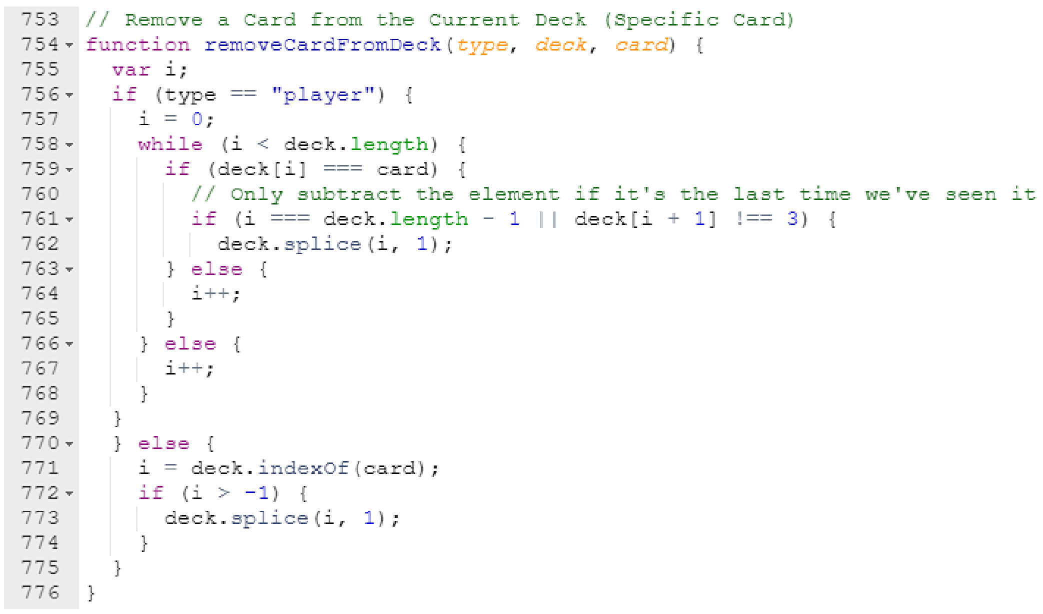
3 b.
3.b.i.

```
// Get Player's Current Deck
playersCardDeck = [getProperty("player1Card1", "image"), getProperty("player1Card2",
"image"), getProperty("player1Card3", "image"), getProperty("player1Card4", "image"),
getProperty("player1Card5", "image"), getProperty("player1Card6", "image"),
getProperty("player1Card7", "image")];

// Draw Card(s)
onEvent("drawCard", "click", function() {
  drawCard("player", 1, undefined);
});

function drawCard(type, amount, currentTurn) {
  for (var i = 0; i < amount; i++) {
    var randomCardChoosen = chooseRandomElement(cards, "all");
    if (type == "player") {
      playersCardDeck = playersCardDeck.concat(randomCardChoosen);
      var playerCardCount = playersCardDeck.length;
      setProperty("playerCardsAmount", "text", playerCardCount + " Cards");
      checkDeckCardsAvailability();
    } else {
      if (currentTurn == 2) {bot2CardDeck = bot2CardDeck.concat(randomCardChoosen);}
      if (currentTurn == 3) {bot3CardDeck = bot3CardDeck.concat(randomCardChoosen);}
      if (currentTurn == 4) {bot4CardDeck = bot4CardDeck.concat(randomCardChoosen);}
    }
  }
}
```

3.b.ii.



```
// Remove a Card from the Current Deck (Specific Card)
function removeCardFromDeck(type, deck, card) {
  var i;
  if (type == "player") {
    i = 0;
    while (i < deck.length) {
      if (deck[i] === card) {
        // Only subtract the element if it's the last time we've seen it
        if (i === deck.length - 1 || deck[i + 1] !== 3) {
          deck.splice(i, 1);
        } else {
          i++;
        }
      } else {
        i++;
      }
    }
  } else {
    i = deck.indexOf(card);
    if (i > -1) {
      deck.splice(i, 1);
    }
  }
}
```

3.b.iii.
In this excerpt of code, playersCardDeck is the name of the list being used in this response.

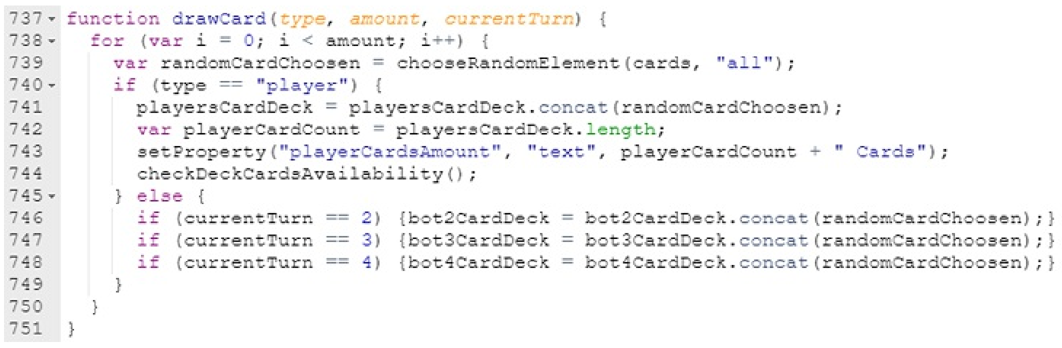
3.b.iv.
The data in the list represents the player's entire card deck. (all cards, initially stored as image IDs.)

3.b.v.

The list in combination with the function manages complexity by allowing a discard from the player's deck in only a few lines, rather than hundreds. Without use of a list it would be very redundant and inefficient to call.

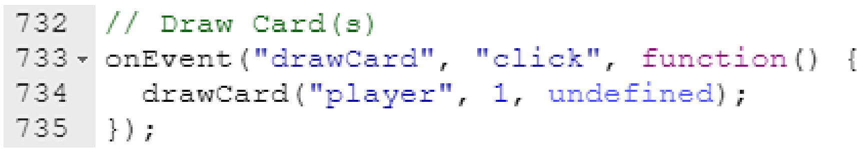
3 c.

3.c.i.

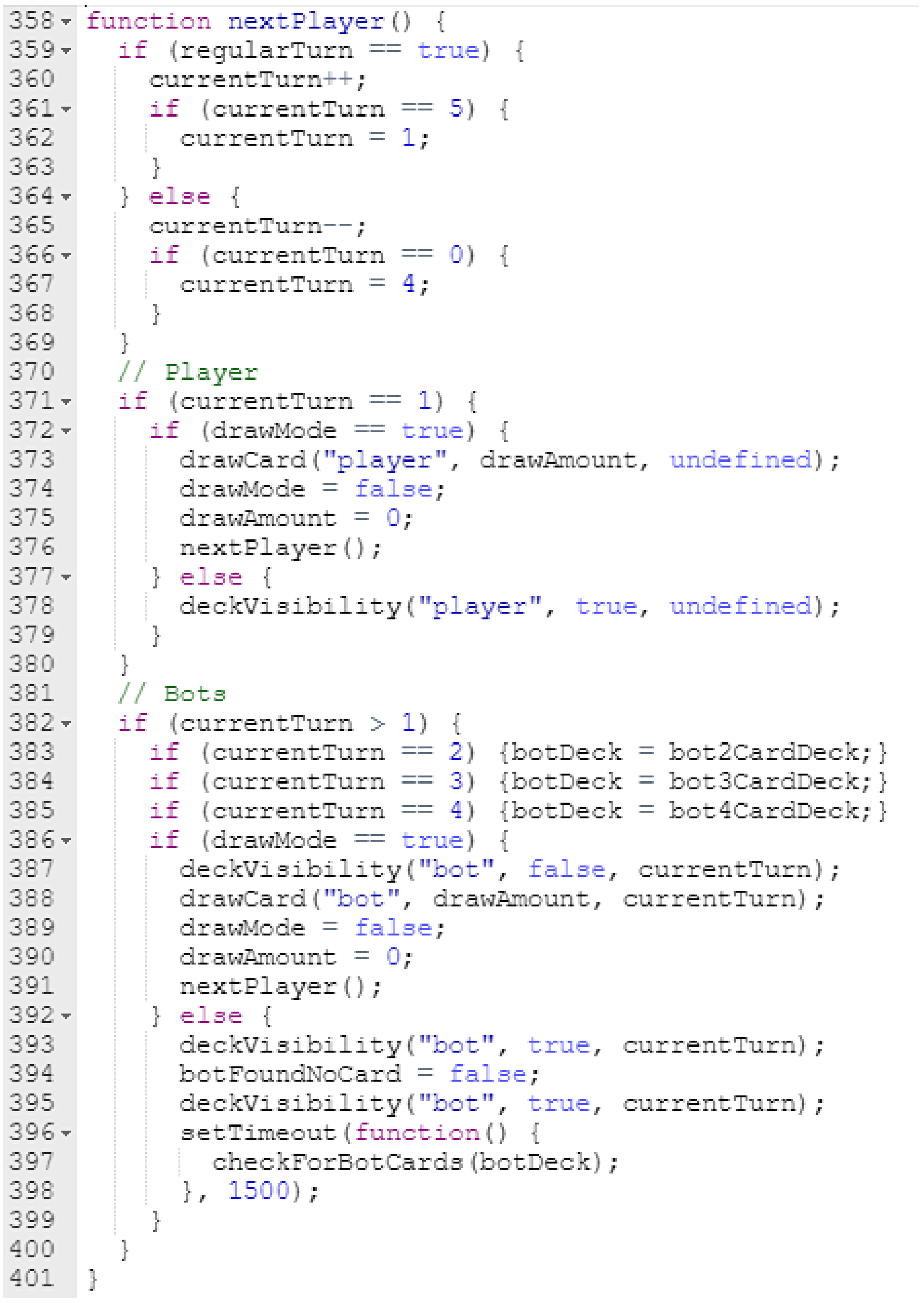


```
function drawCard(type, amount, currentTurn) {
  for (var i = 0; i < amount; i++) {
    var randomCardChoosen = chooseRandomElement(cards, "all");
    if (type == "player") {
      playersCardDeck = playersCardDeck.concat(randomCardChoosen);
      var playerCardCount = playersCardDeck.length;
      setProperty("playerCardsAmount", "text", playerCardCount + " Cards");
      checkDeckCardsAvailability();
    } else {
      if (currentTurn == 2) {bot2CardDeck = bot2CardDeck.concat(randomCardChoosen);}
      if (currentTurn == 3) {bot3CardDeck = bot3CardDeck.concat(randomCardChoosen);}
      if (currentTurn == 4) {bot4CardDeck = bot4CardDeck.concat(randomCardChoosen);}
    }
  }
}
```

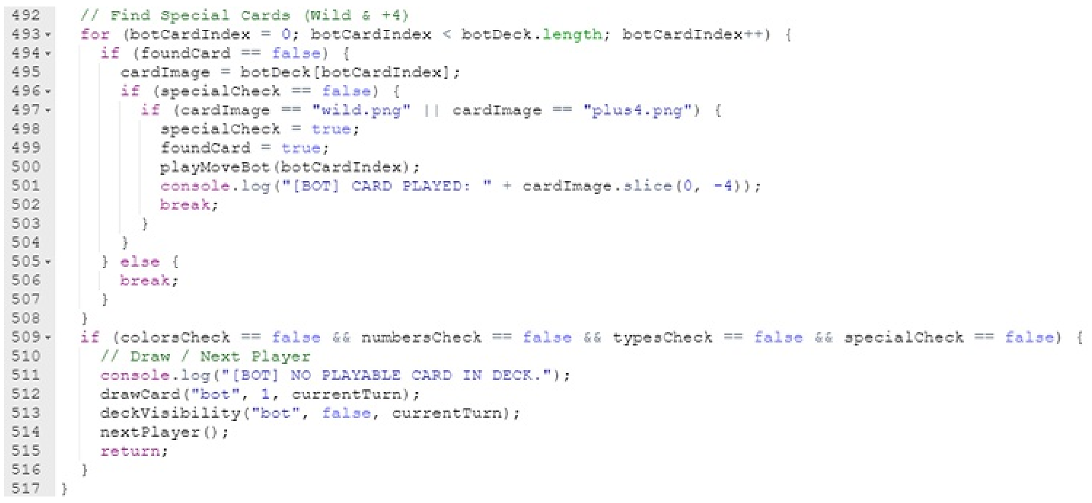
3.c.ii.



```
// Draw Card(s)
onEvent("drawCard", "click", function() {
  drawCard("player", 1, undefined);
});
```



```
function nextPlayer() {
  if (regularTurn == true) {
    currentTurn++;
    if (currentTurn == 5) {
      currentTurn = 1;
    }
  } else {
    currentTurn--;
    if (currentTurn == 0) {
      currentTurn = 4;
    }
  }
  // Player
  if (currentTurn == 1) {
    if (drawMode == true) {
      drawCard("player", drawAmount, undefined);
      drawMode = false;
      drawAmount = 0;
      nextPlayer();
    } else {
      deckVisibility("player", true, undefined);
    }
  }
  // Bots
  if (currentTurn > 1) {
    if (currentTurn == 2) {botDeck = bot2CardDeck;}
    if (currentTurn == 3) {botDeck = bot3CardDeck;}
    if (currentTurn == 4) {botDeck = bot4CardDeck;}
    if (drawMode == true) {
      deckVisibility("bot", false, currentTurn);
      drawCard("bot", drawAmount, currentTurn);
      drawMode = false;
      drawAmount = 0;
      nextPlayer();
    } else {
      deckVisibility("bot", true, currentTurn);
      botFoundNoCard = false;
      deckVisibility("bot", true, currentTurn);
      setTimeout(function() {
        checkForBotCards(botDeck);
      }, 1500);
    }
  }
}
```



```
  // Find Special Cards (Wild & +4)
  for (botCardIndex = 0; botCardIndex < botDeck.length; botCardIndex++) {
    if (foundCard == false) {
      cardImage = botDeck[botCardIndex];
      if (specialCheck == false) {
        if (cardImage == "wild.png" || cardImage == "plus4.png") {
          specialCheck = true;
          foundCard = true;
          playMoveBot(botCardIndex);
          console.log("[BOT] CARD PLAYED: " + cardImage.slice(0, -4));
          break;
        }
      }
    } else {
      break;
    }
  }
  if (colorsCheck == false && numbersCheck == false && typesCheck == false && specialCheck == false) {
    // Draw / Next Player
    console.log("[BOT] NO PLAYABLE CARD IN DECK.");
    drawCard("bot", 1, currentTurn);
    deckVisibility("bot", false, currentTurn);
    nextPlayer();
    return;
  }
}
```

3.c.iii.

The drawCard function adds one card to a specified deck (player or bot), allowing players to progress in gameplay. This allows for turns to be properly cycled and resolves dead-ends or stops in the game.

3.c.iv.

The "drawCard" function begins with iteration, looping the "amount" (function parameter) of times (cards being drawn). A random card is chosen and set to a variable. It then compares the type parameter to determine whether it is a player or bot. The users' deck list is then concatenated with the chosen card. For players, the visual card counter (playersCardAmount) is updated appropriately.

3 d.

3.d.i.

First call:

```
drawCard("player",1,undefined);

Draws one card for the player's card deck.
```

Second call:

```
drawCard("bot",drawAmount,undefined);
```

Draws the drawAmount of cards (dependent on previous card play) for the bot's card deck based on the currentTurn.

3 d.ii.

Condition(s) tested by first call:

The first call tests whether the type argument is equal to the string "player" using the equality operator (==). It also tests whether the amount argument is greater than 0 by checking the loop condition i < amount. (If currentTurn is undefined, then it is read as a player) Thus the player's deck adds a card.

Condition(s) tested by second call:

The second call tests whether the type argument is equal to the string "bot" using the equality operator (==). It also tests the value of the currentTurn argument to determine which bot's card deck to modify. Additionally, it checks whether the drawAmount argument is greater than 0 by checking the loop condition i < amount. Thus adding one card to the bot's deck.

3.d.iii.

Results of the first call:

The player's deck is incremented by a count of one. The list appends a new card value, which is used in other instances throughout the code, thus giving the player one entire additional card.

Results of the second call:

The bot's deck is incremented by a count of drawAmount (in this case that is dependent on a special card in play). The list appends a new card value, which is used in other instances throughout the code, thus giving the bot one entire additional card.