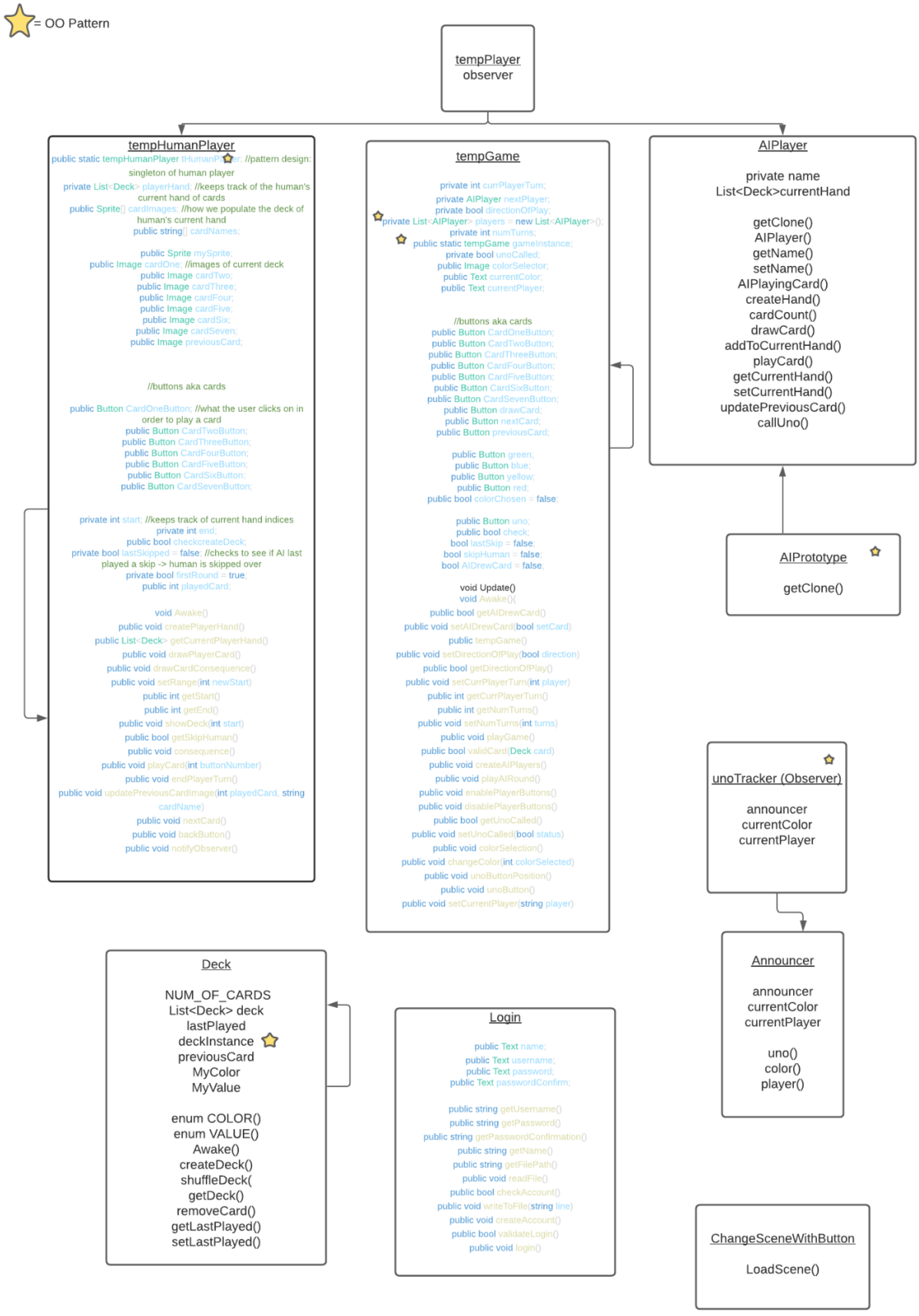
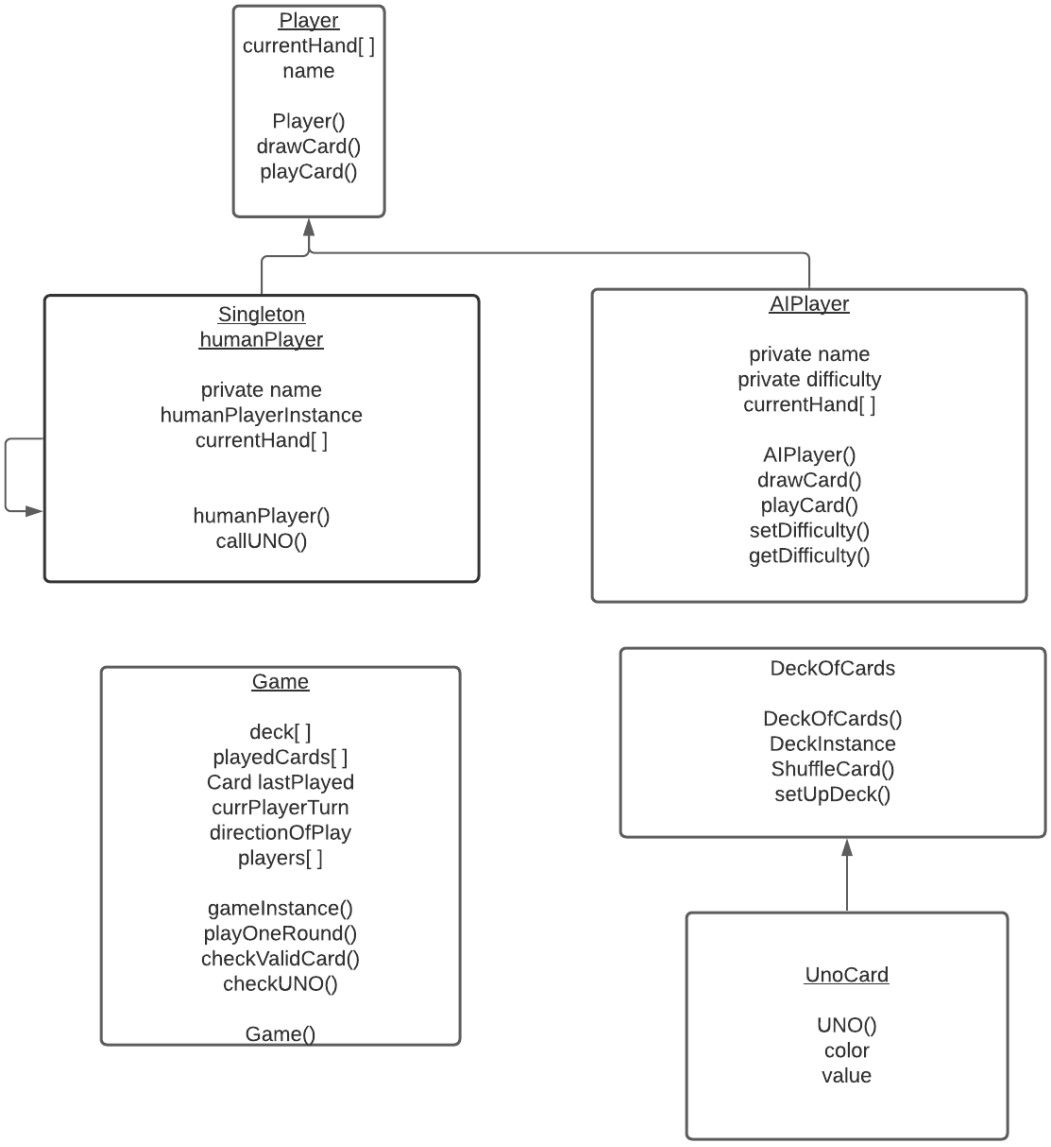
Project 7: Final Project Submission

1. Name of Project: UNO
   Team Members: Seok Jun Song, Emma King
2. Final State of System Statement
    a. The features that we were able to implement in our UNO game were as follows: the draw, shuffle, play, call ("UNO"), view current hand, next card, previous card, AI autonomous play, check card, login, and create account. This means that we were able to build a functioning game that allows users to compete against three AI players and play UNO just like they would against human players. Some features that we have had to move away from include giving the human the ability to pick out different rules, levels of difficulty, and number of AI. We had to scale back the scope of our project because we underestimated the amount of time it would take us to get the visual portion of the game done; previous to this project we had very little experience with Unity and did not realize how steep the learning curve would be. The finalized version of project 7 is different from its previous iterations because we implemented different patterns than we previously planned to. Instead of using the Factory pattern to generate the AI players, we decided to use the Prototype pattern because we had never used it before (unlike Factory) and thought it would be a learning experience. We also decided to implement Observer, so we chose that as our fourth design pattern - the current color and player are displayed on the top right corner of our game. Finally, we chose to implement the Composite pattern as it fit really well with the actions we wanted to have our AI players take. Overall, while we decreased the scope of our project and switched around which design patterns we implemented, we are still quite proud of the finished product as it allows the user to complete a full game of UNO while enjoying the basic actions offered by an in-person UNO game.
3. UML
    a. Current version
        i. We utilized the Observer, Composite, Prototype, and Singleton patterns in our code. We use Observer to show what the current color being played is and which player is currently playing - this is displayed in the top right corner of our game's UI via text. For Composite, we add each of the three AI players into a players list in tempGame and do the same actions to each of them when running through an AI turn. To generate all of our AI players, we used Prototype; this allowed us to clone the very first instance of an AI player created instead of instantiating. Lastly, we used Singleton frequently throughout our code. We created a singular instance of our tempGame, tempHumanPlayer, and deck in order to use them throughout our game.

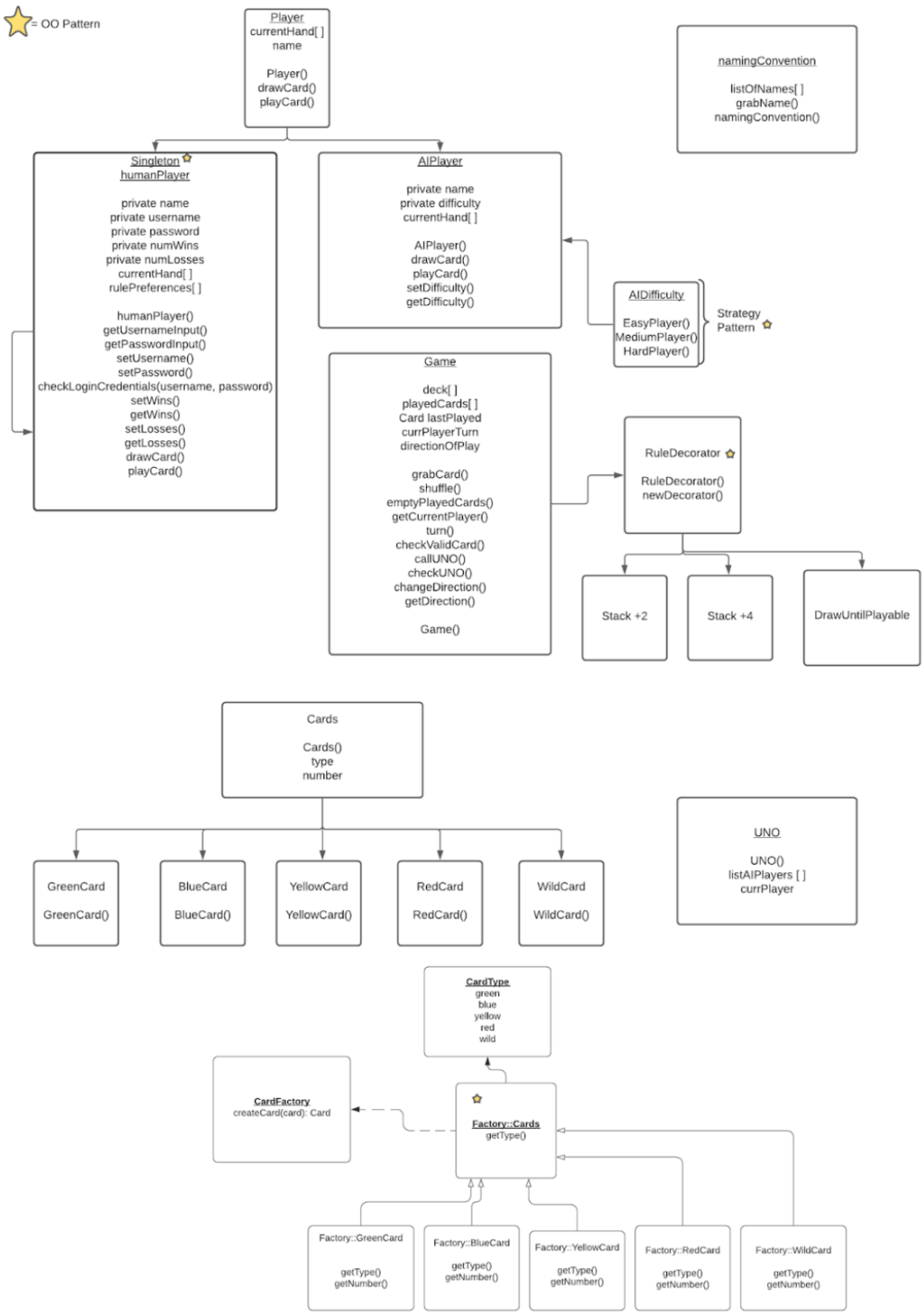

= OO Pattern
tempPlayer
observer
tempHumanPlayer
public static tempHumanPlayer tHumanPlayer; //pattern design: singleton of human player
private List<Deck> playerHand; //keeps track of the human's current hand of cards
public Sprite[] cardImages; //how we populate the deck of human's current hand
public string[] cardNames;
public Sprite mySprite;
public Image cardOne; //images of current deck
public Image cardTwo;
public Image cardThree;
public Image cardFour;
public Image cardFive;
public Image cardSix;
public Image cardSeven;
public Image previousCard;
//buttons aka cards
public Button CardOneButton; //what the user clicks on in order to play a card
public Button CardTwoButton;
public Button CardThreeButton;
public Button CardFourButton;
public Button CardFiveButton;
public Button CardSixButton;
public Button CardSevenButton;
private int start; //keeps track of current hand indices
private int end;
public bool checkcreateDeck;
private bool lastSkipped = false; //checks to see if AI last played a skip -> human is skipped over
private bool firstRound = true;
public int playedCard;
void Awake()
public void createPlayerHand()
public List<Deck> getCurrentPlayerHand()
public void drawPlayerCard()
public void drawCardConsequence()
public void setRange(int newStart)
public int getStart()
public int getEnd()
public void showDeck(int start)
public bool getSkipHuman()
public void consequence()
public void playCard(int buttonNumber)
public void endPlayerTurn()
public void updatePreviousCardImage(int playedCard, string cardName)
public void nextCard()
public void backButton()
public void notifyObserver()
tempGame
private int currPlayerTurn;
private AIPlayer nextPlayer;
private bool directionOfPlay;
private List<AIPlayer> players = new List<AIPlayer>();
private int numTurns;
public static tempGame gameInstance;
private bool unoCalled;
public Image colorSelector;
public Text currentColor;
public Text currentPlayer;
//buttons aka cards
public Button CardOneButton;
public Button CardTwoButton;
public Button CardThreeButton;
public Button CardFourButton;
public Button CardFiveButton;
public Button CardSixButton;
public Button CardSevenButton;
public Button drawCard;
public Button nextCard;
public Button previousCard;
public Button green;
public Button blue;
public Button yellow;
public Button red;
public bool colorChosen = false;
public Button uno;
public bool check;
bool lastSkip = false;
bool skipHuman = false;
bool AIDrewCard = false;
void Update()
void Awake(){
public bool getAIDrewCard()
public void setAIDrewCard(bool setCard)
public tempGame()
public void setDirectionOfPlay(bool direction)
public bool getDirectionOfPlay()
public void setCurrPlayerTurn(int player)
public int getCurrPlayerTurn()
public int getNumTurns()
public void setNumTurns(int turns)
public void playGame()
public bool validCard(Deck card)
public void createAIPlayers()
public void playAIRound()
public void enablePlayerButtons()
public void disablePlayerButtons()
public bool getUnoCalled()
public void setUnoCalled(bool status)
public void colorSelection()
public void changeColor(int colorSelected)
public void unoButtonPosition()
public void unoButton()
public void setCurrentPlayer(string player)
AIPlayer
private name
List<Deck>currentHand
getClone()
AIPlayer()
getName()
setName()
AIPlayingCard()
createHand()
cardCount()
drawCard()
addToCurrentHand()
playCard()
getCurrentHand()
setCurrentHand()
updatePreviousCard()
callUno()
AIPrototype
getClone()
unoTracker (Observer)
announcer
currentColor
currentPlayer
Announcer
announcer
currentColor
currentPlayer
uno()
color()
player()
Deck
NUM_OF_CARDS
List<Deck> deck
lastPlayed
deckInstance
previousCard
MyColor
MyValue
enum COLOR()
enum VALUE()
Awake()
createDeck()
shuffleDeck(
getDeck()
removeCard()
getLastPlayed()
setLastPlayed()
Login
public Text name;
public Text username;
public Text password;
public Text passwordConfirm;
public string getUsername()
public string getPassword()
public string getPasswordConfirmation()
public string getName()
public string getFilePath()
public void readFile()
public bool checkAccount()
public void writeToFile(string line)
public void createAccount()
public bool validateLogin()
public void login()
ChangeSceneWithButton
LoadScene()

b. Project 6 version



i. From project 6 to project 7, we have made several changes to our game. Instead of utilizing a class UnoCard, we downsized to create only a class of Deck. Each instance of Deck was one single UNO card, so our singleton instance of deckInstance was a list of Deck and represented all the available cards for the players to grab. We added in our login class and started using different design patterns than we had originally planned on. We shifted to an AIPrototype interface to implement the Prototype pattern and added in an Observer. In addition, we got rid of the difficulty levels in AIPlayer and added in more variables in the tempGame and tempHumanPlayer to account for occurrences in the game we had not previously planned for and for visual aspects of the UI.

c. Project 5 version



i. We made massive changes between project 5 and project 7. One prominent change regards how we created the cards. In project 5, we used a Factory design pattern that would produce cards based off of their color, whereas in project 7, we used a singular class that uses enumeration to generate the deck of cards. We also had an UNO class in project 5, where we expected to create the overall progressive game logic of the project; however, we pivoted to creating this game logic in the tempGame class instead in our final version. In our earlier versions, we also expected to include the decorator pattern and allow our user to pick and choose which rules they wanted to use in their game. In our later version, we ran out of time to implement the ability to choose different rules so the decorator pattern was never built. In addition, we got rid of the ability to change the AI difficulty level - so all the AI players follow the same logic. We also removed the naming convention as we generically labeled the AI players as "AI1," "AI2," and "AI3." While we did end up implementing the login and creating account classes, which allow the user to create their own account or login to the game, we do not allow the user to see their ratio of wins and losses like previously planned. Overall, we kept most of the basics the same but pivoted on many things due to unforeseen circumstances such as time and ability to get the code working.

4. Third-Party code vs. original code Statement
    a. In order to figure out how the visual aspect of the UNO card displays in Unity, we used the official Unity guide linked below. We also used several other documentations from Unity, but the following link is the base guide that we went off of.
        i. https://docs.unity3d.com/Manual/UnityManual.html
        ii. https://docs.unity3d.com/Manual/Sprites.html
    b. We based our Deck class off of the following link. It was a poker game tutorial that detailed how to create enumerate variables and then create a deck of cards and shuffle those cards based on those enum values. This tutorial provided the basis of the Deck class and taught us how to use enum values.
        i. https://www.youtube.com/watch?v=6SovupmTRt8
    c. We created our prototype design pattern used in the generation of our AI players based off of the example given in the following link.
        i. https://www.javatpoint.com/prototype-design-pattern
    d. We created our composite design pattern used in the actions of our group of AI players based off of the example given in the following link.
        i. https://ronnieschaniel.medium.com/object-oriented-design-patterns-explained-using-practical-examples-84807445b092

    e. We went off of our old code and some in-class notes in order to create the Observer and Singleton patterns utilized in UNO.
5. 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
    a. One key design process issue that occurred during our project was that of the creation of our AI players as we tried a more intuitive design approach instead of tackling the problem with design patterns. In the beginning of this issue, we had a mostly negative experience with it as when creating the AI players as the code was very repetitive, which made us realize that the implementation of a design pattern (such as Prototype) would get us a much better result. While the code looks a bit more complex after implementing the Prototype pattern as we had to include an additional interface, it is now much more robust.
    b. Another key design process element that came up in the creation of our project was when we were implementing the logic behind the AI players. Previous to (and somewhat during) the implementation of the logic, we expected to say “AI1” does this and then “AI2” does that and so on in order to get flesh out the AI turn. We quickly realized that it would be much more efficient to add each AI player into a list and then use the same actions for each one using the Composite pattern. This turned out to be a pretty positive experience and outcome as it allowed our code to become much more flexible and allowed us to add in more AI players with very little code.
    c. One last key design process that we went through was that we needed to flesh out our idea and classes more before starting to implement our idea. While we completed our UML for project 5 and thought that we had covered all the scenarios while going through our plan, we missed several important parts. For example, we had to add in a call (“UNO”) function and a lot more user interface variables/functions than we had been previously anticipating. Overall, we learned that it was more of an agile methodology approach rather than a waterfall type of design process.