Apples to Apples

***Overview:***

Apples to Apples is a card game that can have a varying number of players (more than 2) and some easily customizable rules for playing it. The cards themselves don’t do much, but they do vary in use based on their type, red or green. Every round, a green card is shown with some phrase to be finished, and every player except the “Judge” chooses a red card to complete it. The Judge picks a red card given that they believe best fits the green card’s phrase, then the owner of the red card chosen gets to hold the green card of that round while all red cards used in that round go into a “box,” not to be used again unless needed. After that, the “Judge” position moves to the next player, every player that placed a card(s) draws until they have a certain amount of cards in their hand. The game ends when any player holds a certain number of green cards determined at the beginning of the game.

*Object: Game*

Behaviors:

This object will essentially keep most of the materials and any particular rules for the game in order.

1. When created, Game takes parameters for the amount of players, players’ maximum hand size, and green cards needed to win.
2. Includes arrays for red and green card decks, placements of those cards, and the players.
3. Creates the card set, giving each card beginning or ending phrases depending on the card deck they’ll be housed in.
4. Makes sure that the “Judge” position rotates each round.
5. Makes sure that all players have a certain number of cards in their hand at the end of each round (min 1).
6. Ensures that the program continues until any player collects enough green cards.

Pseudo Code:

CLASS Game( INTEGER playerAmt, INTEGER handSize, INTEGER winCon ){

ASSIGN Array<Card>(playerAmt) to Table[]

ASSIGN Array<Player( ASSIGN handSize to INTEGER hS )>(playerAmt) to players[playerAmt]

ASSIGN Array<Card>(100) to redApples[]

ASSIGN Array<Card>(50) to greenApples[]

ASSIGN Array<Card>( sizeOf( redApples ) ) to redBox[]

ASSIGN winCon to INTEGER wC

ASSIGN 0 to INTEGER judge

ASSIGN 0 to INTEGER rounds

FUNCTION makeApples() //Creates red and green card decks

ASSIGN 0 to INTEGER x

WHILE redApples[sizeOf(redApples)] is equal to null

ASSIGN new Card(STRING text) to redApples[x]

INCREMENT x by 1

ENDWHILE

ASSIGN 0 to x

WHILE greenApples[ sizeOf( greenApples ) ] is equal to null

ASSIGN new Card( STRING text ) to greenApples[x]

INCREMENT x by 1

ENDWHILE

ENDFUNCTION

FUNCTION dealHands()

FOR each player in players[]

players[x].drawNewCard()

ENDFOR

ENDFUNCTION

FUNCTION dealCard( STRING cardType )

emptyDeck()

ASSIGN new Card( STRING text ) to dealt

IF cardType is equal to “green”

FOR each card in greenApples[]

IF greenApples[x].isPlayed() is equal to FALSE

ASSIGN greenApples[x] to dealtCard

greenApples[x].setIsPlayed(

ASSIGN TRUE to BOOLEAN bool)

ENDIF

ENDFOR

ELSEIF cardType is equal to “red”

FOR each card in redApples[]

IF redApples[x].isPlayed() is equal to FALSE

ASSIGN redApples[x] to dealtCard

greenApples[x].setIsPlayed(

ASSIGN TRUE to BOOLEAN bool)

ENDIF

ENDFOR

ENDIF

RETURN dealt

ENDFUNCTION

FUNCTION emptyDeck()

IF redApples[ sizeOf( redApples[] ) ].isPlayed() is equal to TRUE

FOR each Card in redApples[]

If redBox[x] is not equal to null

ASSIGN redBox[x} to redApples[x]

ASSIGN redBox[x} to null

ELSE

BREAK

ENDIF

ENDFOR

ENDIF

IF greenApples[ sizeOf( greenApples[] ) ].isPlayed() is equal to TRUE

FOR each Card in greenApples[]

greenApples[x].setIsPlayed( ASSIGN FALSE to BOOLEAN bool )

ENDFOR

ENDIF

ENDFUNCTION

FUNCTION newJudge()

players[ judge ].setJudge( ASSIGN FALSE to BOOLEAN a )

players[ judge + 1 ].setIsJudge( ASSIGN TRUE to BOOLEAN a )

ENDFUNCTION

FUNCTION gameOver()

ASSIGN FALSE to BOOLEAN win

FOR each player in players[]

IF players[x].getRoundsWon() RETURNS winCon

ASSIGN TRUE to win

ENDIF

ENDFOR

RETURN win

ENDFUNCTION

FUNCTION main()

makeApples()

FOR each player in players[]

players{x].setPlayerNum( ASSIGN x to INTEGER pNum )

ENDFOR

REPEAT

dealHands()

newJudge()

ASSIGN greenApples[ rounds ] to Table[0]

ASSIGN 1 to INTEGER y

FOR each player in players[]

IF players[x].isJudge() is equal to FALSE

playerASSIGN players[x].chooseCard() to Table[y]

INCREMENT y

ENDIF

ENDFOR

FOR each player in players[]

IF players[x].isJudge() is equal to TRUE

ASSIGN players[x].chooseFromTable(

ASSIGN Table[] to Card Table[] )

to INTEGER winnerOfRound

Table[winnerOfRound].getOwner().winRound()

//gets reference to owner of card

ENDIF

ENDFOR

INCREMENT rounds

UNTIL gameOver() RETURNS TRUE

ENDFUNCTION

ENDCLASS

*Object: Player*

Behaviors:

This object represents a person playing the card game.

1. When created, Player takes a single parameter for its maximum hand size (hS) .
2. It holds an array of cards, representing a hand of cards.
3. Every player can become a “Judge” but there’s only one Judge per round.
4. Can choose cards from their ‘hand’ to play
5. Can ‘draw’ new cards
6. When judge, can choose the winning card of the round

Pseudo Code:

CLASS Player( INTEGER hS )

ASSIGN Array<Card>(hS) to Hand[]

ASSIGN 0 to greenCards

ASSIGN FALSE to isJudge

ASSIGN 0 to playerNum

FUNCTION drawNewCard()

ASSIGN reference

FOR each Card in Hand[]

IF Hand[x] is equal to NULL

ASSIGN Game.dealCard() to Hand[x]

Hand[x].setOwner( self )

ENDIF

ENDFOR

ENDFUNCTION

FUNCTION chooseCard()

ASSIGN Int(arc4random\_uniform( sizeOf( Hand[] ) ) to INTEGER choice

ASSIGN Hand[ choice ] to Card chosen

ASSIGN NULL to Hand[x]

RETURN chosen

ENDFUNCTION

FUNCTION chooseFromTable( Card Table[] )

ASSIGN Int( arc4random\_uniform( sizeOf( Table[] ) ) ) to INTEGER choice

RETURN choice

ENDFUNCTION

FUNCTION winRound()

INCREMENT greenCards

ENDFUNCTION

FUNCTION getRoundsWon()

RETURN greenCards

ENDFUNCTION

FUNCTION setPlayerNum( INTEGER pNum )

ASSIGN pNum to playerNum

ENDFUNCTION

FUNCTION getPlayerNum()

RETURN playerNum

ENDFUNCTION

FUNCTION setIsJudge(bool a)

ASSIGN a to isJudge

ENDFUCTION

ENDCLASS

*Object: Card*

Behaviors:

This object represents a card (apple) in this game.

1. When created, Card takes a single parameter for its card text.
2. It holds a reference to its owner so points (green cards) can be awarded accordingly.

Pseudo Code:

CLASS Card( STRING text )

ASSIGN text to STRING phrase

ASSIGN NULL to Player owner

ASSIGN NULL to isPlayed

FUNCTIOIN getText()

RETURN text

ENDFUNCTION

FUNCTION setOwner( Player p )

ASSIGN p to owner

ENDFUNCTION

FUNCTION getOwner()

RETURN owner

ENDFUNCTION

FUNCTION setIsPlayed(BOOLEAN d)

ASSIGN d to isPlayed

ENDFUNCTION

FUNCTION getIsPlayed()

RETURN isPlayed

ENDFUNCTION

ENDCLASS

***Decisions Made:***

As I was designing this, I actually decided to hold off finding how to properly give a score to a player in code because the Judge is supposed to choose a card anonymously. I later realized that reference variables were the way to go since nothing besides a Card object would be able to identify where it came from. However, once I tried to implement this in the pseudo code, I realized it would be slightly more complex than I originally thought. Even now I’m unsure if I implemented those correctly in this pseudo code example.

Even though keeping track of the Cards’ movement is a very important part of the game as far as the condition for winning was concerned, it was better to first deal with the bulk of game (the Game() class), then the Player class as they would naturally interact the most.