Logo

Description automatically generated

Title

**UNO! V.26**

**Card Game**

Course

**CIS-17A**

Section

**43396**

Due Date

**June 6, 2021**

Author

**Lindsay Kislingbury**

**1 Introduction**

Title: UNO!

Uno is a shedding-type card game. The player attempts to get rid of cards in their hand while adding cards to other players’ hands. A player wins when there are no cards left in their hand. There are several “action” cards in UNO which change the play direction, change the color of discard, and add/remove cards from players’ hands.

This version of the game includes a bot with three levels of difficulty to play against. This is accomplished using polymorphism and inheritance.

Github repository: https://github.com/lm2203793/KislingburyLindsay\_CSC17A\_43396/tree/master/Proj/Project2

# **1.1 Summary:**

Project Size: ~1300 lines

This project demonstrates the concepts covered in Chapters 13 through 16:

* Implementation of Inheritance and Polymorphism allowed me to introduce a bot player in this version of the game. The Cplayer and Hplayer classes are inherited from the Player class, making it possible for a Cplayer or Hplayer class object to call different versions of the main game functions.
* Because the Cplayer is inherited from the Player class, a Cplayer object can call Player functions to validate a play, play a card, and change the color of discard without asking for input and using its own logic.
* Operator Overloading is used to print formatted cards with the << operator.
* A Card class copy constructor is used to assign cards with the = operator

This project also demonstrates concepts from earlier chapters:

* Memory is dynamically allocated for the array of Player objects, with allows pointers to derived class Cplayer and Hplayer objects to exist in the same array.
* I am particularly proud of the bot logic in this program:
  + The abstract function getPcrd is overridden in Cplayer to allow the bot to make choices.
  + A vector of valid cards is created by calling the Player class function valCrds and storing the valid cards in a vector.
  + Other Cplayer functions assign values to these valid cards based on their frequency.
  + There are three bot levels: 1, 2 and 3.
  + Bot Level 1 “Dumb Bot” simply chooses a random card from the vector of valid cards.
  + Bot Level 2 “Lucky Bot” flips a coin and chooses the best card based on either the frequency of valid cards that match the symbol (type and number) or the color of discard.
  + Bot Level 3: “Smart Bot” chooses the best card after adding the frequencies of both symbol and color.
    - Occasionally this bot will cause issues (playing the same card repeatedly) when run against another bot. Unfortunately, I did not have the time to fix this issue.

**2. UNO! Rules**

*Official Rules published by Mattel can be found here: https://service.mattel.com/instruction\_sheets/42001pr.pdf*

**Setup:**

The game is for 2-10 players.

Every player starts with 7 cards.

The rest of the cards are placed in the draw pile.

A discard pile is created by flipping over a card from the draw pile.

If the top card is a Wild or Wild Draw 4, it is returned to the deck and another card is flipped.

**Cards:**

108 Cards

|  |  |
| --- | --- |
| Number Cards  19 Blue Cards: 0 to 9  19 Green Cards: 0 to 9  19 Red Cards: 0 to 9  19 Yellow Cards: 0 to 9 | Action Cards:  8 Reverse Cards: 2 each in Blue, Green, Red and Yellow  8 Skip Cards: 2 each in Blue, Green, Red and Yellow  8 Draw-2 Cards: 2 each in Blue, Green, Red and Yellow  4 Wild Cards  4 Wild Draw-4 Cards |

**Game Play:**

* Players examine their card and try to match the top card to the discard.
* Cards are matched by color, number, or action.

For example, if the discard is blue 5, a player has the option of playing any blue card or any color card with a 5.

* Wild cards may be played at any time and the player may choose to the change the leading color with it.
* If the player does not have a playable card, they must draw from the draw pile.
* If the card drawn can be played, the player must play it.

*Note: If the draw pile is exhausted, the draw pile is shuffled and becomes the new draw pile. Play continues on the single card from discard as normal.*

* Play continues until a player has a single card.
* The moment a player has just one card they must call “UNO!” If they do not call “UNO!” before the next player has taken their turn, that player must draw two new cards as penalty.

Calling “UNO!” needs to be repeated every time a player is left with one card.

* Once a player has no cards remaining, the game is over and points are scored.

**Action Cards:**

|  |  |
| --- | --- |
| Reverse | Switch the direction of turns. If the play was moving left, it moves right. |
| Skip | The next player’s turn is skipped. |
| Draw-2 | The next player must draw 2 cards. |
| Wild | This card can be used to represent any color and can be placed on any card.  The player chooses which color it will represent for the next player’s turn. |
| Wild Draw-4 | Acts just like a Wild card except that the next player also has to draw 4 cards. |

**Scoring:**

When a player no longer has any cards the game ends and that player is the winner.

The winner receives points for the cards left in all other players’ hands.

**Points**:

|  |  |  |  |
| --- | --- | --- | --- |
| All Number Cards | Face Value | Reverse | 20 Points |
| Draw-2 | 20 Points | Wild | 50 Points |
| Skip | 20 Points | Wild Draw-4 | 50 Points |

**3 Development**

# **3.1 Class Hierarchy**

**Diagram

Description automatically generated**

**3.2 plyGame Function**

# 3.2.1 plyGame() pseudo code

plyGame is the main game function which mediates between the Deck object and Player objects.

void plyGame()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int numPlyrs, //Number of players

curPlyr, //Current player

cardChc; //Card choice

char cont, //User input to continue after message

unoChc; //Player choice to call uno or play a card

bool unoFlag, //Flag, if player called uno

cPlay, //Flag, if player has a playable card

endgame, //Flag, if game is running endgame=false

trnOver, //Flag, if turn is over=true

error, //Input Validation Error Checking

tryAgn;

Deck deck; //Deck object

Player \*plyrPtr; //Pointer to individual player

Player \*\*plyrs; //Pointer to array of players

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SET tryAgn to true

WHILE tryAgn is true

PRINT "How many Players? (Up to 4): "

INPUT numPlyrs;

IF numPlyrs less than 0 and numPlyrs greater than or equal to 4 and cin is successful

SET tryAgn to false

ELSE

clear buffer

PRINT "Invalid!"

end of while

ALLOCATE memory for array of Player object pointers at plyrs

FOR each Player pointer in plyrs [i=0 to numPlyrs]

CREATE bot choice input variable

SET tryAgn to true

WHILE tryAgn is true

PRINT "Player: Human or Bot? (h or b): "

INPUT bot choice

IF bot choice is b or bot choice is h and cin is successful

SET tryAgn to false

end of if

ELSE

clear buffer

PRINT "Invalid!"

end of else

IF bot choice is h

ALLOCATE memory for Hplayer object

SET pointer in plyrs array to Hplayer object

end of if

ELSE IF bot choice is b

CREATE difficulty level input variable

ALLOCATE memory for Cplayer object

SET pointer in plyrs array to Cplayer object

SET tryAgn to true

WHILE tryAgn is true

PRINT "Difficulty? 1: Dumb Bot 2: Lucky Bot 3: Smart Bot

Enter 1, 2 or 3: "

INPUT difChc

IF tryAgn is less than 0 or tryAgn is greater than 3 or cin fails

clear buffer

PRINT "Invalid!"

end of if

ELSE

SET tryAgn to false

end of else

end of while

SET Cplayer object difficulty level

end of else

CREATE name input variable

PRINT "Enter Player Name: "

clear buffer

INPUT player name

CALL plyrs[i].set(i, name)

FOR 7 times

CREATE Card object temp

CALL and SET temp to deck.dealCrds()

CALL plyrs[i]->recCrd(temp)

end of for loop

DO WHILE endgame is false

SET plyrPtr to plyrs array at curPlyr

CALL plyrPtr->setHnSz()

CALL prcCard(plyrPtr, deck, deck.getDis())

PRINT plyrPtr->getName() "'s Turn!"

CALL and SET cPlay to plyrPtr->cPlyStat(deck.getDis())

CALL plyrPtr->unoStat(cPlay)

CALL deck.getDis()

CALL plyrPtr->showHnd()

IF the current player is an Hplayer object

PRINT "Enter u To Call UNO or any other key to continue: "

INPUT unoChc;

end of if

ELSE

IF player has uno

SET unoChc to u

end of if

ELSE

clear buffer

PRINT <<plyrPtr->getName()<<" doesn't call uno."<<endl; //msg

end of else

end of else

CALL uno(plyrPtr, deck, unoChc, cPlay)

DO WHILE trnOver is false

clear buffer

IF cPlay is true

CALL deck.chngDisc(plyrPtr->getPcrd(deck.getDis()))

SET trnOver to true

end of if

ELSE IF cPlay is false

DO WHILE cPlay is false

CREATE validate and SET to false

IF the current player is a Hplayer object

PRINT "No Valid Card to Play!"

PRINT Enter any key to continue"

INPUT cont

end of if

ELSE

PRINT plyrPtr->getName()

"has no valid card to play!"

end of else

CALL plyrPtr->recCrd(deck.dealCrds())

CALL and SET validate to plyrPtr->valLstCrd(deck.getDis())

PRINT plyrPtr->getName() " Drew: "

PRINT plyrPtr->shoLstCrd()

IF validate is true

IF the current player is a Hplayer object

PRINT "Enter any key to Play it"

clear buffer

INPUT cont

end of if

CALL deck.chngDisc(plyrPtr->playTop())

SET cPlay to true

SET trnOver to true

end of if

end of do while

end of else

IF hand size is 0

CALL plyrPtr->setWin()

FOR each player [i=0 to numPlyrs]

CALL plyrs[i]->setScr()

end of for loop

PRINT <"END GAME!!"

PRINT plyrPtr->getName() " WINS!!!!"

FOR each player [i=0 to numPlyrs]

CALL plyrs[i]->prntScr()

end of for loop

SET endgame to true

end of if

end of do while

CALL and SET curPlyr to setPlyr(deck.getDis(), curPlyr, numPlyrs)

end do while

CALL wrtScr(plyrs, numPlyrs)

DELETE []plyrs

# Diagram Description automatically generated3.2.1 plyGame() flowchart

*Diagram

Description automatically generated*

*Diagram

Description automatically generated*

*Diagram

Description automatically generated*

# 3.3 getPcrd Function

Polymorphic behavior is only possible when objects are passed by reference to the abstract function.

By using a pointer to the player objects in the main game function, Cplayer and Hplayer objects can call different versions of the getPcrd function.

3.3.1 Cplayer getPcrd pseudo code

This version of the getPcrd function calculates a “value” for each card in the players hand based on the frequency of colors and symbols (number and type) in the hand.

After running a few tests, I determined that all three bot levels won more often when they prioritized DRAW2, REVERSE, and SKIP types, and de-prioritized WILD and WILD4 types.

**Card Cplayer::getPcrd(Card &disCrd)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Card temp; //Card object to return**

**vector<int> valCrds; //Valid card indexes**

**int vecSz=0; //Size of valid card vector**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**FOR i=0 to hand.size()**

**IF valPlay(disCrd,hand[i])) returns true**

**CALL valCrds.push\_back(i)**

**increment vector size**

**end of if**

**end of for loop**

**SWITCH on hardNum**

**case 1**

**CREATE index**

**SET index to random element in valCrds**

**SET temp to hand[valCrds[index]]**

**CALL hand.erase(hand.begin()+valCrds[index])**

**end of case 1**

**case 2**

**CREATE coin**

**SET coin to random number 1 or 2**

**IF coin is 1**

**CREATE sScrs[14] and initalize all elements to 0**

**CALL scoreSym(sScrs) calculate valid card values based on symbol**

**CREATE smax variable to hold index of card with highest value**

**CALL and SET smax to scrCrdsN(valCrds, sScrs, vecSz)**

**CREATE index**

**SET index to valCrds[smax]**

**SET temp to hand[index]**

**CREATE and SET erase to valCrds[smax]**

**CALL hand.erase(hand.begin()+erase)**

**end of if**

**ELSE**

**CREATE cScrs[4] and initalize all elements to 0**

**CALL scoreClr(cScrs) calculate valid card values based on color**

**CREATE cmax variable to hold index of card with highest value**

**CALL and SET cmax to scrCrdsC(valCrds, cScrs, vecSz)**

**CREATE index**

**SET index to valCrds[cmax]**

**SET temp to hand[index]**

**CREATE AND SET erase to valCrds[cmax]**

**CALL hand.erase(hand.begin()+erase)**

**end of case 2**

**case 3**

**CREATE sScrs[14] and initalize all elements to 0**

**CALL scoreSym(sScrs) calculate valid card values based on symbol**

**CREATE cScrs[4] and initalize all elements to 0**

**CALL scoreClr(cScrs) calculate valid card values based on color**

**CREATE amax variable to hold index of card with highest value**

**CALL and SET amax to sCrdsA(sScrs, cScrs, valCrds, vecSz)**

**SET index to valCrds[amax]**

**SET temp to hand[index]**

**CREATE AND SET erase to valCrds[amax]**

**CALL hand.erase(hand.begin()+erase)**

**end of case 3**

**end of switch**

**IF temp.getType() is WILD or temp.getType() is WILD4)**

**CREATE wCol**

**CALL and SET wCol to chsWild()**

**RETURN wCol**

**CALL temp.setColor(wCol)**

**end of if**

**PRINT name " plays "**

**PRINT temp**

**RETURN temp**

**3.3.1.1 Supporting Functions for Cplayer::getPcrd**

**3.3.1.2 scoreClr**

void Cplayer::scoreClr(int cScrs[])

FOR each Card in hand

CREATE and SET color to hand[i].getColor()

SWITCH on color

case r : increment cScrs[0]

case g : increment cScrs[1]

case b : increment cScrs[2]

case y : increment cScrs[3]

end of switch

**3.3.1.3 scrCrdsC**

int Cplayer::scrCrdsC(vector<int> valCrds, int cScrs[], int vecSz){

CREATE colTots[vecSz] and initialize all elements to 0

FOR each element i in valCrds

CREATE and SET col to hand[valCrds[i]].getColor();

SWITCH on color

case r : SET colTots[i] to cScrs[0]

case g : SET colTots[i] to cScrs[1]

case b : SET colTots[i] to cScrs[2]

case y : SET colTots[i] to cScrs[3]

case X : SET colTots[i] to 0 de

end of switch

CREATE cmax and SET to 0

FOR each element i in vecSz

IF colTots[i] is greater than cmax)

SET cmax to i

end of if

end of for loop

RETURN cmax

3.2.1 Hplayer getPcrd pseudo code

The Hplayer version of getPcrd gets input from the user and utilizes an exception class to perform input validation.

Card Hplayer::getPcrd(Card &disCrd)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int chc; //Hold card choice

Card temp; //Temp Card object to return

int index; //Index of card choice for temp card

bool tryAgn=true; //Input validation flag

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

clear buffer

PRINT "What card do you want to play?"

INPUT chc

WHILE tryAgn is true

TRY

CALL and SET index to pCrd(disCrd, chc)

SET tryAgn to false;

end of try

CATCH(BadChc bad)

PRINT "Error: " bad.getChc() " is an invalid choice!"

PRINT "Enter the number that corresponds to your card choice: "

INPUT chc

end of catch

end of while

CREATE colChc

IF hand[index].getType() is WILD or hand[index].getType() is WILD4

CALL and SET colChc to chsWild()

CALL and SET colChc to tolower(colChc)

CALL hand[index].setColor(colChc)

end of if

SET temp to hand[index]

CALL hand.erase(hand.begin()+index)

RETURN temp

**4. Project Check Off Sheet**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Chapter** | **Section** | **Topic** | **Where Line #''s** | **Pts** | **Notes** | | 13 |  | Classes |  |  |  | |  | 1 to 3 | Instance of a Class | (main) 84 | 4 | The Deck object deck is declared here. The classes Card, Deck, Player, Hplayer and Cplayer are defined in their respective .h files | |  | 4 | Private Data Members | (Player.h) 18-24 | 4 | Never Public | |  | 5 | Specification vs. Implementation | Player: Player.h  Player.cpp Cplayer: Cplayer.h  Cplayer.cpp Hplayer: Hplayer.h  Hplayer.cpp | 4 | .h vs. .cpp files | |  | 6 | Inline | (Player.h) 52 | 4 | setHnSz() adds the current hand size to the hndSzs  vector in one line within the Player.h file | |  | 7, 8, 10 | Constructors | (Card.h) 44, (Deck.cpp) 14 | 4 | The members of Card are set in it's Constructor. When new cards are added to the deck, their member values are passed to the Constructor | |  | 9 | Destructors |  | 4 |  | |  | 12 | Arrays of Objects | Declared (main) 86,  Filled (main 16 & 131) | 4 | Array of Player pointers holds pointers to derived  class Cplayer and Hplayer objects | |  | 16 | UML | (in Project 2 Folder)  UML.png | 4 |  | |  |  |  |  |  |  | | 14 |  | More about Classes |  |  |  | |  | 1 | Static |  | 5 |  | |  | 2 | Friends | (Card.h) 76 | 2 | Overloaded << operator is a friend of ostream class | |  | 4 | Copy Constructors | (Card.h) 36-41 | 5 | Overloads the = operator | |  | 5 | Operator Overloading | << (Card.h) 76 = (Card.h) 35 | 8 | Overload 3 operators | |  | 7 | Aggregation | (Deck.h) 17-19 | 6 | The Deck class has Card class member variables | |  |  |  |  |  |  | | 15 |  | Inheritance |  |  |  | |  | 1 | Protected members | (Player.h) 18-24 | 6 | The protected Player members are accessed by derived classes Cplayer and Hplayer. | |  | 2 to 5 | Base Class to Derived | Player.h, Cplayer.h, and Hplayer.h | 6 | The Cplayer and Hplayer classes are derived  from the Player class | |  | 6 | Polymorphic associations | (Player.h) 32 (Cplayer.h) 26 (Hplayer.h) 24 | 6 | The abstract member function getPcrd is overriden  in the derived classes Cplayer and Hplayer | |  | 7 | Abstract Classes |  | 6 |  | |  |  |  |  |  |  | | 16 |  | Advanced Classes |  |  |  | |  | 1 | Exceptions | (Hplayer.cpp) 27-36 | 6 | Exception class BadChc is used for input validation | |  | 2 to 4 | Templates |  | 6 |  | |  | 5 | STL | (Deck.cpp) 82-83 | 6 | STL vector member function are used extensively  throughout the program. Here, back() and pop\_back() are used | |  |  |  |  |  |  | |  |  | Sum |  | 100 |  | |