```
"""
*********************************************************************************
 FILE:      Game

 AUTHOR:    Peter Campbell

 ASSIGNMENT: Laboratory 6

 DATE:      11/26/2012

 DESCRIPTION: This program creates the game cribbage.

 *********************************************************************************
"""

from cs110graphics import *
import random

#initiates and creates methods for the card class
class Card(Group, EventHandler):
    def __init__(self, game, suit, rank, value, faceFileName, backFileName):
        Group.__init__(self)
        EventHandler.__init__(self)
        self._suit = suit
        self._value = value
        self._rank = rank
        self._player = None
        self._game = game
        self._faceImage = Image(faceFileName)
        self._backImage = Image(backFileName)
        self._selected = False

        # initially, the card is face down:
        self._upImage = self._backImage
        self._downImage = self._faceImage
        self.add(self._upImage)
        self.add(self._downImage)
        self.addHandler(self)

    def handleMouseRelease(self, button, loc):
        """handles the mouse release on a card depending on whose turn it is and
    what part of the game it is"""
        if self._game.getCurrentPlayer() == 0:
            hand = self._game.getPlayerA().getHand()
        elif self._game.getCurrentPlayer() == 1:
            hand = self._game.getPlayerB().getHand()

        if self in hand:
            if self._game.getGamePlay() == 'crib':
                self._game.cribSelection(self)
            elif self._game.getGamePlay() == 'count':
                self._game.countSelect(self)

    def setPlayer(self, player):
        self._player = player

    def getPlayer(self):
        return self._player

    def getRank(self):
        return self._rank

    def getValue(self):
```

```
64          return self._value
65
66      def getSuit(self):
67          return self._suit
68
69      def size(self):
70          return self._upImage.size()
71
72      def isSelected(self):
73          return self._selected
74
75      def flip(self):
76          self.remove(self._upImage)
77          self.remove(self._downImage)
78          self._upImage, self._downImage = self._downImage, self._upImage
79          self.add(self._upImage)
80          self.add(self._downImage)
81
82      def changeDepth(self, depth):
83          self.setDepth(depth)
84
85  #initiates and creates methods for the deck class
86  class Deck(Group):
87      def __init__(self, game):
88          Group.__init__(self)
89          self._deckImage = Image('Cards/back.gif', Point(50, 225))
90          self._deckImage.setDepth(30)
91          self.add(self._deckImage)
92
93          #rank is used to order the cards so 11 is a jack 12 is a queen and 13 is
94          #a king. Value is the point value of that card ie face cards are 10
95          rankNames = ['1', '2', '3', '4', '5', '6', '7', '8', '9',
96                       '10', '11', '12', '13']
97          valueNames = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
98          suitNames = ['clubs', 'diamonds', 'hearts', 'spades']
99          folderName = 'Cards/'
100         backName = 'Cards/back.gif'
101         self._game = game
102         self._cards = []
103         for suit in suitNames:
104             for i in range(len(rankNames)):
105                 card = Card(self._game, suit, rankNames[i], valueNames[i],
106                             folderName + suit[0] + rankNames[i] + '.gif',
107                             backName)
108                 self._cards.append(card)
109
110     def deal(self):
111         card = self._cards.pop()
112         return card
113
114     def shuffle(self):
115         random.shuffle(self._cards)
116
117
118 #initiates and creates methods for the Board class
119 class Board(Group):
120     def __init__(self):
121         Group.__init__(self)
122         #self._pegA1
123         #self._pegA2
124         #self._pegB1
125         #self._pegB2
126         self._holeCentersA = []
```

```
127             self._holeCentersB = []
128
129             #The following code adds the holes to the board in order so that the
130             #pegs can move to their center points depending on how many points the
131             #player has.
132             for i in range(36):
133                 self._holeCentersA.append(Point(200 + (22 * i), 340))
134             for i in range(36):
135                 self._holeCentersB.append(Point(200 + (22 * i), 363))
136             self._holeCentersA = self._holeCentersA + [Point(1008, 342),
137                                                        Point(1053, 362),
138                                                        Point(1088, 393),
139                                                        Point(1098, 434),
140                                                        Point(1098, 460),
141                                                        Point(1088, 501),
142                                                        Point(1053, 532),
143                                                        Point(1008, 552)]
144             self._holeCentersB = self._holeCentersB + [Point(1006, 365),
145                                                        Point(1041, 380),
146                                                        Point(1069, 405),
147                                                        Point(1076, 434),
148                                                        Point(1076, 460),
149                                                        Point(1069, 489),
150                                                        Point(1041, 514),
151                                                        Point(1006, 529)]
152             for i in range(36):
153                 self._holeCentersA.append(Point(200 + (22 * i), 553))
154             for i in range(36):
155                 self._holeCentersB.append(Point(200 + (22 * i), 530))
156             self._holeCentersA = self._holeCentersA + [Point(170, 550),
157                                                        Point(142, 526),
158                                                        Point(130, 494),
159                                                        Point(142, 462),
160                                                        Point(170, 438)]
161             self._holeCentersB = self._holeCentersB + [Point(178, 530),
162                                                        Point(162, 515),
163                                                        Point(152, 494),
164                                                        Point(162, 473),
165                                                        Point(178, 458)]
166             for i in range(35):
167                 self._holeCentersA.append(Point(200 + (22 * i), 435))
168             for i in range(35):
169                 self._holeCentersB.append(Point(200 + (22 * i), 458))
170             self._holeCentersA.append(Point(1015, 447))
171             self._holeCentersB.append(Point(1015, 447))
172
173             #creates the image of the board itself and its circles
174             self._board = Image('Cards/boardnc.jpg', Point(600, 450))
175             self._board.resize(1100, 300)
176             self._board.setDepth(40)
177             for i in range(len(self._holeCentersA)):
178                 circle = Circle(7, self._holeCentersA[i])
179                 circle.setDepth(30)
180                 circle.setFillColor('black')
181                 self.add(circle)
182             for i in range(len(self._holeCentersB)):
183                 circle = Circle(7, self._holeCentersB[i])
184                 circle.setDepth(30)
185                 circle.setFillColor('black')
186                 self.add(circle)
187             self.add(self._board)
188
189     #initiates and creates methods for the ReadyButton class
```

```
190 class ReadyButton(Group, EventHandler):
191     def __init__(self, game):
192         Group.__init__(self)
193         EventHandler.__init__(self)
194         self._game = game
195         self._button = Square(80, Point(0, 0))
196         self._button.setFillColor('gold')
197         self.add(self._button)
198         self._txt = Text('Discard', Point(0, 0))
199         self._go = False
200         self._timesGoClicked = 0
201         self._txt.setDepth(30)
202         self.add(self._txt)
203         self.setDepth(40)
204         self.addHandler(self)
205 
206     #handles to mouse releas of the button depending on what the current
207     #gameplay is and what the button is being used for at that time
208     def handleMouseRelease(self, button, loc):
209         if self._game.getGamePlay() == 'crib':
210             self._game.moveToCrib()
211         elif self._go:
212             #the second click of go give a point to the player who clicked
213             if self._timesGoClicked == 1:
214                 self._game.clearCounting()
215                 self._game._countingScore = 0
216                 self._timesGoClicked = 0
217                 if self._game.getCurrentPlayer() == 0:
218                     self._game.getPlayerA().addScore(1)
219                 else:
220                     self._game.getPlayerB().addScore(1)
221                 self.update()
222                 self._game.changePlayer()
223             else:
224                 self.update()
225                 self._game.changePlayer()
226                 self._timesGoClicked += 1
227         else: #self._game.getGamePlay() == 'count':
228             self._timesGoClicked = 0
229             self._game.counting()
230 
231 
232 
233 
234     def update(self):
235         """Changes the button depending on gameplay and the cards in players
236     hands during counting"""
237         if self._game.getGamePlay() == 'crib':
238             if self._game.getCurrentPlayer() == 0:
239                 self._game.getReadyButton().moveTo(Point(200, 100))
240             elif self._game.getCurrentPlayer() == 1:
241                 self._txt.setTextString('Play')
242         elif self._game.getGamePlay() == 'count':
243             if self._game.getCurrentPlayer() == 0:
244                 self._game.getReadyButton().moveTo(Point(200, 100))
245                 self._txt.setTextString('Go!')
246                 self._go = True
247                 for card in self._game.getPlayerB().getHand():
248                     if card.getValue() + self._game.getCountingScore() <= 31:
249                         self._txt.setTextString('Play')
250                         self._timesGoClicked = 0
251                         self._go = False
252             elif self._game.getCurrentPlayer() == 1:
```

```
253                 self._game.getReadyButton().moveTo(Point(700, 700))
254                 self._txt.setTextString('Go!')
255                 self._go = True
256                 for card in self._game.getPlayerA().getHand():
257                     if card.getValue() + self._game.getCountingScore() <= 31:
258                         self._txt.setTextString('Play')
259                         self._timesGoClicked = 0
260                         self._go = False
261
262 #initiates and creates methods for the Player class
263 class Player:
264     def __init__(self, name, loc):
265         self._name = name
266         self._loc = loc
267         self._score = 0
268         self._txt = Text(str(self._name + ':' + str(self._score)), self._loc,
269                          25)
270         self._txt.setDepth(30)
271         self._txt.setTextColor('black')
272         self._hand = []
273
274     def addTo(self, container):
275         container.add(self._txt)
276
277     def addCard(self, card):
278         self._hand.append(card)
279
280     def removeCard(self, card):
281         self._hand.remove(card)
282
283     def getHand(self):
284         return self._hand
285
286     def assign(self, lst):
287         self._hand = lst
288         #return self._hand
289
290     def addScore(self, points):
291         self._score = self._score + points
292         self._txt.setTextString(str(self._name + ':' + str(self._score)))
293
294     def getScore(self):
295         return self._score
296
297 def sortCards(cards):
298     newCards = []
299     for i in range(13):
300         for card in cards:
301             if int(card.getRank()) == i + 1:
302                 newCards.append(card)
303     return newCards
304
305 #This is the main class that contains the different types of gameplay that will
306 #be determined by eventhandlers.
307 class Game(Window):
308     def __init__(self):
309         Window.__init__(self, 1300, 800, 'darkgreen', 'Cribbage')
310         self._currentPlayer = 0
311         self._currentCrib = 0
312         self._currentGamePlay = 'crib'
313         self._cribImage = None
314         self._countingScore = 0
315         self._countScoreTxt = Text("The count is: " + str(self._countingScore),
```

```
316                                     Point(900, 225), 25)
317         self.add(self._countScoreTxt)
318
319         board = Board()
320         self.add(board)
321
322         self._deck = Deck(self)
323         self.add(self._deck)
324         self._deck.shuffle()
325
326         self._readyButton = ReadyButton(self)
327         self._readyButton.moveTo(Point(900, 700))
328         self.add(self._readyButton)
329
330         #creates a list of the cards being counted and if the counting cards are
331         #cleared they are dumoed into discard
332         self._countingCards = []
333         self._countingDiscard = []
334         self._starterCard = None
335
336         self._playerA = Player('PlayerA', Point(75, 700))
337         self._playerA.addTo(self)
338         self._playerB = Player('PlayerB', Point(75, 100))
339         self._playerB.addTo(self)
340
341         self._player = 1
342
343         for i in range(6):
344             self._playerA.addCard(self._deck.deal())
345             self._playerA.getHand()[i].setPlayer(0)
346         self._playerA.assign(sortCards(self._playerA.getHand()))
347         self._width, _ = self._playerA.getHand()[0].size()
348
349         for i in range(6):
350             self._playerA.getHand()[i].move(250 + self._width * (i + 1), 680)
351             self.add(self._playerA.getHand()[i])
352
353         for i in range(6):
354             self._playerB.getHand().append(self._deck.deal())
355             self._playerB.getHand()[i].setPlayer(1)
356         self._playerB.assign(sortCards(self._playerB.getHand()))
357
358         for i in range(6):
359             self._playerB.getHand()[i].move(250 + self._width * (i + 1), 100)
360             self.add(self._playerB.getHand()[i])
361
362         for i in range(6):
363             self._playerA.getHand()[i].flip()
364
365     def addCount(self, points):
366         self._countingScore = self._countingScore + points
367         self._countScoreTxt.setTextString("The count is: " + \
368                                           str(self._countingScore))
369
370     def getCountingScore(self):
371         return self._countingScore
372
373     def getCurrentPlayer(self):
374         return self._currentPlayer
375
376     def getReadyButton(self):
377         return self._readyButton
378
```

```
379     def getPlayerA(self):
380         return self._playerA
381
382     def getPlayerB(self):
383         return self._playerB
384
385     def changePlayer(self):
386         #flip cards over
387         if self._currentPlayer == 0:
388             hand1 = self._playerA.getHand()
389             hand2 = self._playerB.getHand()
390         elif self._currentPlayer == 1:
391             hand1 = self._playerB.getHand()
392             hand2 = self._playerA.getHand()
393
394         for i in range(len(hand1)):
395             hand1[i].flip()
396         for i in range(len(hand2)):
397             hand2[i].flip()
398
399         self._currentPlayer = (self._currentPlayer + 1) % 2
400
401     def getGamePlay(self):
402         return self._currentGamePlay
403
404     def changeGamePlay(self):
405         if self._currentGamePlay == 'crib':
406             self._currentGamePlay = 'count'
407         elif self._currentGamePlay == 'count':
408             self._currentGamePlay = 'score'
409             self.scoring()
410         elif self._currentGamePlay == 'score':
411             self._currentGamePlay = 'crib'
412
413
414     #crib selection
415     def cribSelection(self, card):
416         numberSelected = 0
417
418         #checks whose turn it is and changes hand appropriately
419         if self._currentPlayer == 0:
420             hand = self._playerA.getHand()
421         elif self._currentPlayer == 1:
422             hand = self._playerB.getHand()
423
424         for i in range(6):
425             if hand[i].isSelected():
426                 numberSelected += 1
427         if numberSelected < 2 and not card.isSelected():
428             card.move(0, -10)
429             card._selected = True
430             numberSelected += 1
431         elif card.isSelected():
432             card.move(0, 10)
433             card._selected = False
434         else:
435             print "You already selected two cards"
436             sleep(1)
437             print ''
438
439
440     def moveToCrib(self):
441         if self._currentPlayer == 0:
```

```
442             hand = self._playerA.getHand()
443         elif self._currentPlayer == 1:
444             hand = self._playerB.getHand()
445         crib = []
446         cribIndex = []
447         for i in range(6):
448             if hand[i].isSelected():
449                 crib.append(hand[i])
450                 cribIndex.append(i)
451 
452         #Need to remove the higher inedexed card first so that the lower indexed
453         #card card still has the same index
454         #CITE: Carson
455         #DETAILS:
456         self.remove(hand[cribIndex[1]])
457         hand.remove(hand[cribIndex[1]])
458         self.remove(hand[cribIndex[0]])
459         hand.remove(hand[cribIndex[0]])
460 
461         self.rearrangeCards(hand)
462         if self._currentPlayer == 0:
463             self._cribImage = Image('Cards/back.gif', Point(700, 225))
464             self.add(self._cribImage)
465             self._readyButton.update()
466             self.changePlayer()
467         elif self._currentPlayer == 1:
468             self._readyButton.update()
469             self.changeGamePlay()
470 
471 
472     #Rearrange cards so there are no gaps
473     def rearrangeCards(self, hand):
474         if hand == sortCards(self._playerA.getHand()):
475             for i in range(len(hand)):
476                 hand[i].moveTo(Point(250 + self._width * (i + 1), 680))
477             self._playerA.assign(hand)
478         elif hand == sortCards(self._playerB.getHand()):
479             for i in range(len(hand)):
480                 hand[i].moveTo(Point(250 + self._width * (i + 1), 100))
481             self._playerB.assign(hand)
482 
483 
484     #from card handler
485     def countSelect(self, card):
486         #Play a card in the middle of the board counts to 31 and adds score when
487         #applicable and moves player pegs"
488         numberSelected = 0
489         if self._currentPlayer == 0:
490             hand = self._playerA.getHand()
491         elif self._currentPlayer == 1:
492             hand = self._playerB.getHand()
493 
494         for i in range(len(hand)):
495             if hand[i].isSelected():
496                 numberSelected += 1
497         if numberSelected < 1 and  not card.isSelected():
498             card.move(0, -10)
499             card._selected = True
500             numberSelected += 1
501         elif card.isSelected():
502             card.move(0, 10)
503             card._selected = False
504 
```

```
505         #from readyButton handler
506         def counting(self):
507             leng = len(self._countingCards)
508             if self._currentPlayer == 0:
509                 for card in self._playerA.getHand():
510                     if card.isSelected():
511                         if card.getValue() + self._countingScore <= 31:
512                             card.moveTo(Point(200 + (leng * self._width / 2), 225))
513                             self._countingCards.append(card)
514                             self.addCount(card.getValue())
515                             if self._countingScore == 31:
516                                 self._playerA.addScore(2)
517                                 self.clearCounting()
518                             self._playerA.removeCard(card)
519                             self.rearrangeCards(self._playerA.getHand())
520                             if len(self._playerB.getHand()) == 0 and \
521                                     len(self._playerA.getHand()) == 0:
522                                 self._playerA.addScore(1)
523                                 self.clearCounting()
524                                 self.changeGamePlay()
525                                 return
526                             elif len(self._playerB.getHand()) == 0:
527                                 self.changePlayer()
528                                 self._readyButton.update()
529                                 self.changePlayer()
530                                 return
531                             else:
532                                 self._readyButton.update()
533                                 self.changePlayer()
534                                 return
535                         else:
536                             print 'Invalid, that card would make the count over 31'
537                             sleep(1.5)
538                             print ''
539
540             elif self._currentPlayer == 1:
541                 for card in self._playerB.getHand():
542                     if card.isSelected():
543                         if card.getValue() + self._countingScore <= 31:
544                             card.moveTo(Point(200 + (leng * self._width / 2), 225))
545                             self._countingCards.append(card)
546                             self.addCount(card.getValue())
547                             if self._countingScore == 31:
548                                 self._playerB.addScore(2)
549                                 self.clearCounting()
550                             self._playerB.removeCard(card)
551                             self.rearrangeCards(self._playerB.getHand())
552                             if len(self._playerA.getHand()) == 0 and \
553                                     len(self._playerB.getHand()) == 0:
554                                 self._playerB.addScore(1)
555                                 self.clearCounting()
556                                 sleep(1)
557                                 self.changeGamePlay()
558                             elif len(self._playerA.getHand()) == 0:
559                                 self.changePlayer()
560                                 self._readyButton.update()
561                                 self.changePlayer()
562                                 return
563                             else:
564                                 self._readyButton.update()
565                                 self.changePlayer()
566                                 return
567                         else:
```

```
568                         print 'Invalid, that card would make the count over 31'
569                         sleep(1.5)
570                         print ''
571
572     def clearCounting(self):
573         for card in self._countingCards:
574             card.moveTo(Point(-100, -100))
575             self._countingDiscard.append(card)
576         self.addCount(-self._countingScore)
577         self._countingCards = []
578         self._readyButton.update()
579
580     def scoring(self):
581         #Puts original cards back into players hands
582         self._readyButton.moveTo(Point(-100, -100))
583         self._starterCard = self._deck.deal()
584         self.add(self._starterCard)
585         self._starterCard.flip()
586         self._starterCard.moveTo(Point(150, 225))
587         sleep(1)
588
589         if self._starterCard.getRank() == 11:
590             if self._currentCrib == 0:
591                 self._playerA.addScore(2)
592             elif self._currentCrib == 1:
593                 self._playerB.addScore(2)
594         for i in range(len(self._countingDiscard)):
595             if self._countingDiscard[i].getPlayer() == 0:
596                 self._playerA.addCard(self._countingDiscard[i])
597             elif self._countingDiscard[i].getPlayer() == 1:
598                 self._playerB.addCard(self._countingDiscard[i])
599         self.rearrangeCards(sortCards(self._playerA.getHand()))
600         self.rearrangeCards(sortCards(self._playerB.getHand()))
601
602         if self._currentCrib ==  0:
603             self._playerB.addCard(self._starterCard)
604             self.rearrangeCards(sortCards(self._playerB.getHand()))
605             print
606             self._playerB.addScore(self.count(self._playerB.getHand()))
607
608         #def countCrib(self):
609
610
611     def count(self, hand):
612         total = 0
613         #check for nobs
614         for i in range(5):
615             if hand[i].getRank() == 11 and hand[i].getSuit() == \
616                     self._starterCard.getSuit():
617                 total = total + 1
618         print str(total)
619         sleep(1)
620
621         #check for pairs
622         for i in range(4):
623             if hand[i].getRank() == hand[i+1].getRank():
624                 total = total + 2
625         print 'pairs ' + str(total)
626         sleep(2)
627
628         #check for 15's
629         for i in range(4):
630             if hand[0].getValue() + hand[i+1].getValue() == 15:
```

```
631                     total = total + 2
632             for i in range(3):
633                 if hand[1].getValue() + hand[i+2].getValue() == 15:
634                     total = total + 2
635             for i in range(2):
636                 if hand[2].getValue() + hand[i+3].getValue() == 15:
637                     total = total + 2
638             if hand[0].getValue() + hand[4].getValue() == 15:
639                 total = total + 2
640 
641             for i in range(3):
642                 if hand[0].getValue() + hand[1].getValue() + \
643                         hand[i+2].getValue() == 15:
644                     total = total + 2
645             for i in range(2):
646                 if hand[0].getValue() + hand[2].getValue() + \
647                         hand[i+3].getValue() == 15:
648                     total = total + 2
649             if hand[0].getValue() + hand[3].getValue() + hand[4].getValue() \
650                     == 15:
651                 total = total + 2
652             for i in range(2):
653                 if hand[1].getValue() + hand[2].getValue() + \
654                         hand[i+3].getValue() == 15:
655                     total = total + 2
656             if hand[2].getValue() + hand[3].getValue() + hand[4].getValue() \
657                     == 15:
658                 total = total + 2
659 
660             print '15 ' + str(total)
661             sleep(2)
662 
663             #check for flush
664             clubs = 0
665             diamonds = 0
666             hearts = 0
667             spades = 0
668             for i in range(5):
669                 if hand[i].getSuit() == 'clubs':
670                     clubs += 1
671                 elif hand[i].getSuit() == 'diamonds':
672                     diamonds += 1
673                 elif hand[i].getSuit() == 'hearts':
674                     hearts += 1
675                 elif hand[i].getSuit() == 'spades':
676                     spades += 1
677             if clubs >= 4 or diamonds >= 4 or hearts >= 4 or spades >= 4:
678                 total = total + 4
679             print 'flush ' + str(total)
680             sleep(2)
681 
682             #check for runs
683         #   for i in range(5):
684         #       for j in range(4):
685         #           if hand[i].getValue()
686 
687             return total
688 
689 
690 def test():
691     Game()
692 
693 if __name__ == "__main__":
```

```
694         StartGraphicsSystem(test)
```