

Правила игры Cow 006

Кто такая Корова 006?

Ее придумал один из известнейших разработчиков настольных игр Вольфганг Крамер. На его счету более двухсот настольных логических игр и он известен не только на просторах родной Германии, но и по всему миру.

Игра рассчитана на игроков в возрасте от 8-ми лет. Принимать участие могут от 2-х до 10-ти человек.

Файл cow006_card.py

Логический класс карты, содержит ее номер и штрафные очки

```
def __init__(self, n):
    self.n = n
    self.opened = True
```

по номеру карты задает штрафные очки карты

```
if self.n % 10 == 0:
    score = 3
elif self.n % 5 == 0:
    score = 2
elif self.n > 10 and len(set(str(self.n))) == 1:
    score = 5

if self.n == 55:
    score = 7

self.score = score
```

Определяет номер карты, её штрафные очки и владельца карты и его штрафные очки

```
def __repr__(self):
    return 'Card n={} (score={})'.format(self.n, self.score)
```

Возвращает данные в виде:

```
Resolve Card n=48 (score=1) from света: 0
Resolve Card n=74 (score=1) from William: 3
Resolve Card n=25 (score=2) from света: 7
```

сравнивает значение карты с остальными

```
def __lt__(self, other):  
    return self.n < other.n
```

исключает карты одного и того же значения

```
def __eq__(self, other):  
    try:  
        return self.n == other.n  
    except AttributeError:  
        return False
```

@staticmethod

```
def all_cards(maxsize=104):  
    """  
        Создание карт для колоды, статический метод  
    """  
    return [Card(i+1) for i in range(maxsize)]
```

Файл cow006_container.py

```
import random
```

```
from cow006_card import Card
```

```
class Container:
```

```
    """
```

Контейнер для карт, реализует базовые методы для работы с ними

```
    """
```

```
    def __init__(self):
```

```
        self.cards = []
```

Добавляет карту на руку

```
    def add_card(self, c):
```

```
        self.cards.append(c)
```

self.cards.pop() выкидывает последний элемент листа

```
    def draw(self):
```

```
        return self.cards.pop()
```

Рандомная перетасовка карт

```
    def shuffle(self):
```

```
        random.shuffle(self.cards)
```

Возвращает количество элементов в контейнере

```
def __len__(self):  
    return len(self.cards)
```

доступ по номеру карты

```
def __getitem__(self, i):  
    return self.cards[i]
```

Это однозначное представление объекта в виде строки, которое можно использовать, чтобы воссоздать точно такой же объект, а если это невозможно, то вывести какое-нибудь полезное сообщение

```
def __repr__(self):  
    return "
```

```
print(deck)
```

Файл cow006_deck.py

```
class Deck:
```

```
    """
```

```
        Класс колоды
```

```
    """
```

```
def __init__(self):  
    self.cards = Card.all_cards()
```

перетасовка карт в колоде

```
def shuffle(self):  
    shuffle(self.cards)
```

выкидывает последний элемент из колоды

```
def draw(self):  
    return self.cards.pop()
```

```
def __str__(self):  
    return '\n'.join([str(card) for card in self.cards])
```

Файл cow006_table.py

```
import random
```

```
from cow006_card import Card
```

```
from cow006_container import Container
```

Задаёт ряд и максимальное количество карт в нём

```
class Row(Container):
```

```
    def __init__(self, card, maxinrow = 6):
```

```
        print('card for row = ', card)
```

```
        self.cards = [card]
```

```
        self.maxinrow = maxinrow      # какая корова "проваливает" ряд
```

```
    def __repr__(self):
```

```
        return 'Row with cards: ' + ' '.join(map(str, self.cards))
```

```
    def __lt__(self, other):
```

```
        pass
```

```
    def top(self):
```

```
        """ возвращает последнюю карту в ряду """
```

```
        return self.cards[-1]
```

```
    def overflow(self):
```

```
        """ проверяет, есть 6 коров в ряду (True) или еще нет (False) """
```

```
        return len(self.cards) >= self.maxinrow
```

```
    def acceptable(self, card):
```

```
        """ эту карту card можно положить в конец этого ряда? """
```

```
        return self.cards[-1].n < card.n
```

```
    def cut(self):
```

```
        """ Убирает из ряда все карты, кроме последней. Возвращает список  
убранных карт """
```

```
        cards = self.cards[:-1]
```

```
        last_card = self.cards[-1]
```

```
        self.cards.clear()
```

```
        self.cards.append(last_card)
```

```
        return cards
```

```
    def get_score(self):
```

```
        return sum([card.score for card in self.cards])
```

```
class Table:
```

```
    def __init__(self, deck, rows = 4, maxinrow = 6):
```

```

self.maxinrow = maxinrow
self.rows = [Row(deck.draw(), maxinrow) for r in range(rows)]

def __repr__(self):
    return '\n'.join(['Row{} : {}'.format(i, r) for i, r in enumerate(self.rows)] )

def find_row(self, card):
    """ ищет, в какие ряды можно положить эту карту, возвращает ряд или
None,
если карту нельзя положить ни в один ряд """

    #index = random.randint(0, 3)
    index = -1
    res_row = None
    diff = 1000000
    for i, row in enumerate(self.rows):
        d = card.n - row.top().n
        if d > 0 and d < diff:
            index = i
            res_row = row
            diff = d

    return res_row, index

def __getitem__(self, i):
    return self.rows[i]

def get_index(self, row):
    for i, r in enumerate(self.rows):
        if r == row:
            return i

```

Файл base_wiget.py

Содержит класс BaseWidget, который наследуется из QWidget. Это базовый виджет для главного экрана игры.

Метод `reset(self, layout)`:

Метод, который удаляет все дочерние виджеты.

Файл `card_widget.py`

Импортирован класс `BaseWidget`.

Содержит класс `Card`, наследуемый из `Qwidget`. Это виджет для карты.

Метод `init(self, card, options={}, clickable=False)`

Метод, который задает размеры карты, размеры пикселей и кликабельность карты.

Метод `init_ui(self)`

Проверяет, открыта ли карта (if `card.opened`), если да то устанавливаются необходимые виджеты, вроде номера карты и штрафа. Далее определяются размеры карты и ее размеры при нажатии (больше на 6 пикселей). Если карта кликабельна, то применяем метод `define_standard_size`.

Метод `set_card(self, card)`

Это метод, который связывает объект карты и виджета для отображения данных

Метод `define_standard_size(cls, width, height)`

Это метод класса, который переопределяет размеры карты.

Файл `arena_widget.py`

Содержит класс `ArenaWidget`, наследуемый из `BaseWidget`. Это виджет-контейнер с областью выкладки карт рубашкой вверх в начале игры.

Метод `init(self, players=None)`

Конструктор класса.

Метод `init_ui(self)`

Создает виджеты, нужные для основного виджета.

Метод `add_card(self, player, card)`

Добавляет виджет карты в область по игроку и логическому объекту карты.

Метод `remove_card(self, player)`

Удаляет карту из контейнера игрока.

Метод `open_all(self)`

Переворачивает карты в контейнере рубашкой вниз

Метод `set_players(self, players)`

Связывание объекта игроков и данного виджета для отображения данных.

Файл `cow006_player.py`

Задаем имена игроков:

```
PLAYER_NAMES = [  
    'John',  
    'Bob',  
    'Adam',  
    'Alice',  
    'Olivia',  
    'Amanda',  
    'Portia',  
    'Charlie',  
    'William',  
    'Sophie',  
    'Mia',  
    'Isabella',  
    'Emily'  
]
```

```
class Player:
```

```
    """
```

```
    # Класс логики игрока: аргумент self ссылается на экземпляр класса для которого  
    вызывается метод. В методе __init__ self ссылается на только что созданный объект.
```

```
    """
```

```
    def __init__(self, name, is_bot=True):
```

```
        self.name = name
```

```
        self.hand = []
```

```
        self.score = 0
```

```
        self.is_bot = is_bot
```

```
    def init_hand(self, deck, maxhand):
```

```
        """
```

```
        # Инициализация карт игрока
```

```
        """
```

```

self.hand.clear()
for i in range(maxhand):
    card = deck.draw()
    self.hand.append(card)

def add_score(self, cards):
    """
    # Добавление штрафных очков игроку с полученных карт
    """
    self.score += sum([card.score for card in cards])

def add_card(self, card):
    """
    # Добавление карты
    """
    self.hand.append(card)

def choose_row(self, table, card):
    """
    # Выбор (для бота) ряда с наименьшим кол-вом штрафных очков
    при невозможности положить карту
    """
    min_score = 100000
    min_index = -1
    for i, row in enumerate(table):
        score = row.get_score()
        if score < min_score:
            min_score = score
            min_index = i
    return table[min_index]

def choose_card(self, table):
    """
    # Выбор (для бота) карты, которую он положит рубашкой вверх.
    Здесь можно реализовать более сложную логику.
    """
    card = self.hand.pop()
    return card

def __str__(self):
    return self.name + ': ' + str(self.score)

```



```
from cow006_player import Player
from cow006_table import Table, Row
from cow006_deck import Deck
```

LIMIT_SCORE = 66

```
class Game:
```

◆◆◆◆◆

#Главный класс игры. В нем происходит логика игры и взаимодействие других логических классов

●●●●●

```
def init (self, players, rows=4, maxinrow=6, maxhand=10):
```

```
self.players = players
```

```
self.maxhand = maxhand
```

из колоды раздают карты на стол и игрокам, больше она не нужна

```
deck = Deck()
```

```
#print(deck)
```

deck.shuffle()

```
self.table = Table(deck, rows, maxinrow)
```

```
for p in self.players:
```

```
p.init_hand(deck, maxhand)
```

```
def run(self):
```

!!!!!!

#Генератор для получения игровых состояний и данных и проброс их в виджеты

◆◆◆◆◆

$$\text{chosen_cards} = \{\}$$

пока не закончатся карты на руке

```
for step in range(self.maxhand):
```

```
# все игроки кладут закрытые карты
```

```
for player in self.players:
```

```
if player.is_bot:
```

```
card = player.choose card(self.table)
```

else:

```
card = yield 'HUMAN SELECT CARD', {}
```

```

    choosen_cards[card] = player
    data = {
        'player': player,
        'card': card
    }
    yield 'PLAYER_PLACE_CARD', data

yield 'OPEN_ALL_CARDS', {}

# карты открываются и выкладываются на стол
for card, player in sorted(choosen_cards.items()):
    print('Resolve {} from {}'.format(card, player))
    row, index = self.table.find_row(card)

    if row is None:
        if player.is_bot:
            # игрок выбирает, в какой ряд положить
            r = player.choose_row(self.table, card)
        else:
            data = {
                'player': player,
                'card': card,
            }
            r = yield 'HUMAN_SELECT_ROW', data
            # кладет карту в конец ряда
            r.add_card(card)
            # забирает все карты, кроме последней
            cutted = r.cut()
            # карты, что забрал игрок, добавляются на его счет
            player.add_score(cutted)
            data = {
                'player': player,
                'card': card,
                'row_index': self.table.get_index(r)
            }
            yield 'PLAYER_CANNOT_PLACE', data
        else:
            row.add_card(card)
            data = {
                'player': player,
                'card': card,
                'row_index': index

```

```

    }
    yield 'PLAYER_MOVE_TO_ROW', data

    # это шестая карта?
    if row.overflow():
        # забираем карты, кроме последней и добавляем их в счет игрока
        cutted = row.cut()
        player.add_score(cutted)
        yield 'PLAYER_CARD_OVERFLOW', data

    choosen_cards.clear()

    state, data = self.end()
    yield state, data

def end(self):
    """
    #Обработка окончания функции run и принятие решения
    об окончании всей игры или только тура
    """
    winlist = self.players[:]
    winlist.sort(key=lambda item: item.score)
    max_score = winlist[-1].score

    if max_score < LIMIT_SCORE:
        return 'NEW_TOUR', {}
    else:
        return 'FINISH', {'winlist': winlist}

```