8. Блэкджек

Постановка задачи

Блэкджек – простая карточная игра. В игру играют двое – игрок и дилер. В игре каждая карта имеет свою стоимость. Задачей игрока является набрать больше очков, чем у дилера.

В блэкджек играют колодой из 52 карт (в каждой масти карты 2, … , 10, валет, дама, король и туз). У каждой карты есть своя стоимость: от 2 до 10 – соответственно 2, … , 10 очков, у картинок (валет, дама и король) – 10 очков, у туза – 1 или 11 очков по желанию игрока.

В начале игры колода перемешивается. Игроку и дилеру изначально раздают по 2 карты, одна карта дилера открыта, а другая – закрыта.

Игрок смотрит свои карты и может попросить ещё карту (англ. hit) или остановиться (англ. stand). Если сумма очков игрока превысит 21, то говорят, что произошёл перебор (англ. bust) – в этом случае игрок сразу проигрывает.

После того как игрок остановился, дилер открывает свою закрытую карту. Если очки дилера меньше 17, он набирает карты, пока его очки не превысят 17. Если сумма очков дилера превысит 21, он проигрывает.

Выигрывает тот, кто набрал больше очков, не превысив при этом 21. В нашей версии при равенстве очков выигрывает дилер.

Для упрощения работы рекомендуется использовать шаблон memory_template.py.

Пример игры

Начало игры

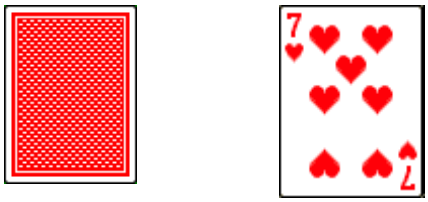
Колода перемешана. Игрокам раздают по две карты. Одна карта дилера закрыта.

Рис. 7.1. Карты в начале игры

Ходы игрока

Сумма очков игрока – 13 (валет – 10 очков, тройка – 3 очка). Игрок решает взять ещё карту.

Карты дилера



Карты игрока

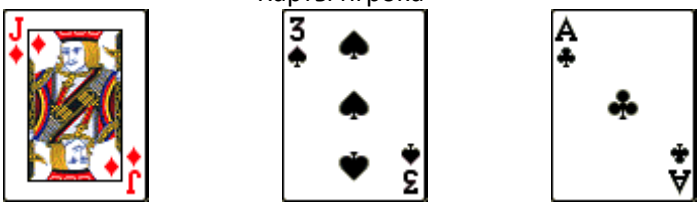
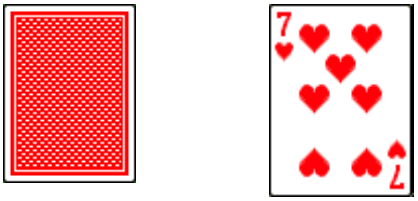


Рис. 7.2. Карты после первого выбора игрока

Игрок может использовать туз только как 1 очко (иначе он получит $10 + 3 + 11 = 24$ очка и проиграет). Сумма очков игрока становится $10 + 3 + 1 = 14$. Игрок решает взять ещё одну карту.

Карты дилера



Карты игрока

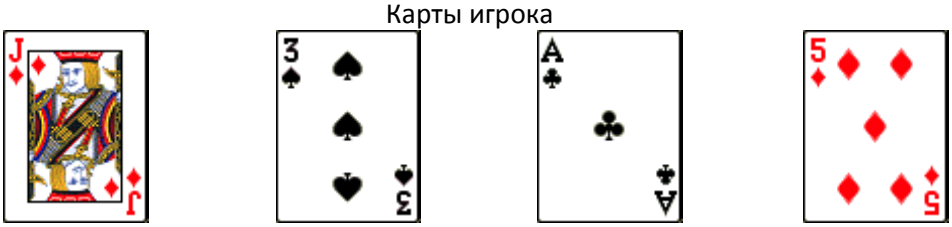


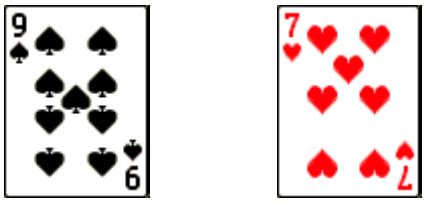
Рис. 7.3. Карты после второго выбора игрока

Сумма очков игрока достигла 19. Игрок решает остановиться.

Ходы дилера

Дилер открывает свою закрытую карту.

Карты дилера



Карты игрока

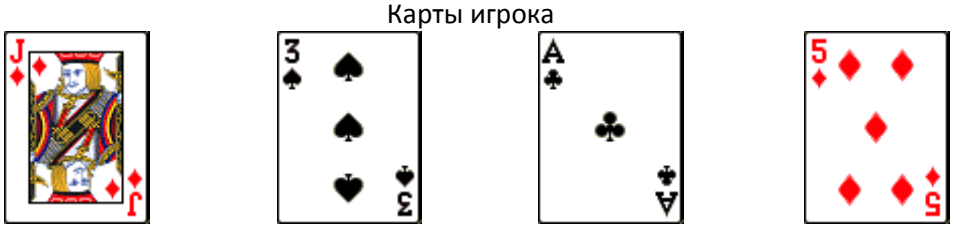
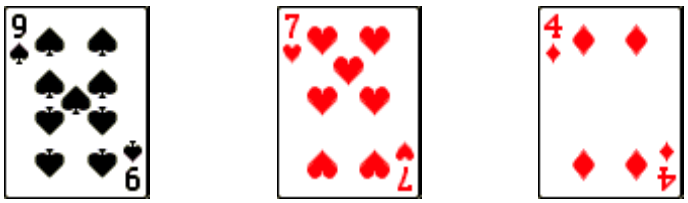


Рис. 7.4. Карты в начале хода дилера

Сумма очков дилера: $7 + 9 = 16$. Так как у дилера меньше 17 очков, он должен взять ещё одну карту.

Карты дилера



Карты игрока

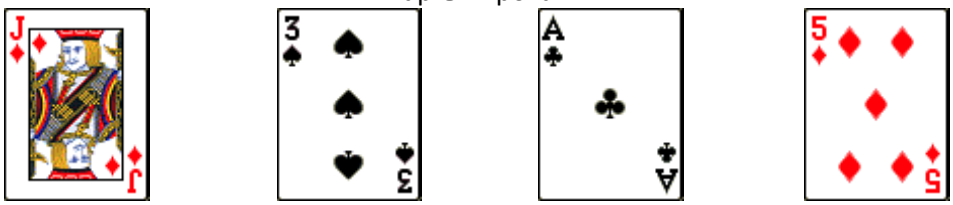


Рис. 7.5. Карты в конце ходов дилера

Сумма очков дилера: $9 + 7 + 4 = 20$. Дилер набрал ≥ 17 очков и должен остановиться.

Определение победителя

У игрока 19 очков. У дилера 20 очков. У дилера больше, чем у игрока, дилер выигрывает.

Перебор

Допустим, на последнем шаге дилер получит даму (рис. 7.6).

В этом случае сумма очков дилера составит $9 + 7 + 10 = 26$ очков, что больше 21. Дилер проигрывает.

Рис. 7.6. Перебор у дилера

Наблюдения за ходом игры

Обратите внимание на следующие моменты.

- У игрока есть два действия – взять ещё карту (англ. hit) и остановиться (англ. stand).
- При показе игрового поля во время хода игрока закрыта одна карта дилера. После того, как игрок остановится, эта карта открывается.
- Обеим сторонам выгодно использовать стоимость туза как 11, до тех пор, пока полученная в результате сумма не превысит 21. После этого туза можно использовать только как 1. Поэтому стоимость туза зависит не от произвола игрока, а от наличия других карт.
- Второго туза всегда можно использовать только как 1 (сумма очков за два туза: $11 + 11 = 22 > 21$). Как 11 может быть засчитан только один туз.

Классы для игры «Блэкджек»

Рассмотрим основные объекты, манипуляции с которыми происходят в игре блэкджек:

- игральные карты (англ. card);
- колода (англ. deck) – набор карт, откуда их выдают;
- рука (англ. hand) – набор карт, выданных игроку или дилеру.

В предложенном шаблоне для каждого из этих объектов создан класс. В качестве имени классов были использованы термины английского языка.

Для представления одной игральной карты используется класс `Card`. В карточных играх существенной информацией, идентифицирующей карту, являются её масть и достоинство (тройка, семёрка, туз, и т.п.). Соответственно поля данных этого класса включают `suit` – масть и `rank` – достоинство[1]. Для вывода карты на экран могут потребоваться и другие поля, специфичные

[1] Для представления мастей в шаблонах игры и тестов используются латинские буквы: C – кресты (♣), S – пики (♠), H – черви (♥), D – буби (♦). Для обозначения достоинств используются латинские буквы: A – туз, T – десятка, J – валет, Q – дама, K – король. Остальные достоинства обозначаются соответствующими цифрами (не числами!).

именно для компьютерной модели карты, такие, как изображение лица (в этом проекте такие поля не будут использоваться).

Игральная карта не имеет сложного поведения, которое меняло бы её внутреннее состояние. Поэтому в предложенном варианте у класса Card есть методы для узнавания масти и достоинства, а также метод для рисования карты на холсте.

Содержанием классов Deck и Hand является набор карт. С точки зрения содержания эти классы можно было бы объединить (например, под названием «набор карт»). Однако они обладают существенно различающимся поведением. Поэтому лучше реализовать эти классы отдельно.

Класс Deck (колода) должен обеспечить методы для перемешивания колоды и выдачи очередной карты (при выдаче карта удаляется из колоды). При создании объекта этого класса его конструктор должен заполнить объект полным набором карт.

Класс Hand (рука) содержит методы для добавления карты и определения суммы очков. При создании объекта он должен быть инициализирован пустым списком карт. В классе Hand также требуется метод для рисования.

В таблице 7.1 представлена сводка по классам для игры «Блэкджек», их данным и методам.

Таблица 7.1. Классы для игры «Блэкджек»

Класс	Данные	Методы
Card	`suit` – масть; `rank` – достоинство.	`get_suit()` – узнать масть, `get_rank()` – узнать достоинство, `draw(canvas, pos)` – нарисовать карту на холсте в заданной позиции.
Deck	Набор карт.	`shuffle()` – перемешать, `deal_card()` - выдать карту.
Hand	Набор карт.	`add_card()` – добавить карту, `get_value()` – узнать сумму очков, `draw(canvas, pos)` – нарисовать на холсте в заданной позиции.

При проектировании системы классов стараются разделять задачи между классами таким образом, чтобы максимально обеспечить возможность повторного использования классов в других проектах. Например, класс Card может быть использован без изменений в любой карточной игре с 52-карточной колодой.

Заметим, что класс Card никак не участвует в подсчёте очков – за это полностью отвечает класс Hand. Это разумно, так как подсчёт очков – свойство конкретной игры, в других играх стоимость карт может быть другой, или её может вообще не быть. Если бы подсчёт очков был помещён в классе Card, то его пришлось бы перерабатывать для использования в других играх.

Кроме того, в игре «Блэкджек» стоимость туза зависит от других карт руки. Это также свидетельствует о том, что помещать расчёт стоимости следует в классе Hand.

Класс Hand в данном проекте привязан к конкретной игре – он обеспечивает хранение карт и подсчёт очков по правилам блэкджек. С этой точки зрения его правильнее называть «рука для блэкджек», а не просто «рука». Для реализации других карточных игр придётся разрабатывать похожий класс, который может отличаться методами и логикой работы.

Не следует слишком увлекаться «универсализацией» классов. Например, класс `Card` можно сделать более универсальным, добавив возможность использовать расширенные наборы достоинств (например, джокер) или наоборот только сокращенную колоду из 36 карт. Однако вполне может быть, что использовать этот класс для других игр никогда не потребуется. Тогда работа, потраченная на реализацию этих функций, пропадёт зря, а использование такого класса в проекте «Блэкджек» окажется немного сложнее (надо будет выбрать правильную колоду).

Поэтому, определяя функции для классов в проекте, ориентируйтесь на те потребности, которые реально есть в существующем проекте. Чтобы понять, в каком классе должна быть реализована каждая функция, подумайте о других возможных использованиях каждого класса. Для более подробного знакомства с принципами объектно-ориентированного дизайна можно обратиться к работе [10].

Рекомендованный порядок работы над заданием

Рекомендуется разбить работу над реализацией игры на две части. В первой части сосредоточьтесь на реализации логики игры. Для вывода информации используйте консоль. Графический интерфейс лучше добавить во второй части.

Реализация логики игры

1. Откройте шаблон blackjack_template.py и ознакомьтесь с классом `Card` (карта). Этот класс уже реализован, Ваша задача – изучить его интерфейс и код. Скопируйте реализацию класса `Card` в шаблон card_test.py и проверьте, что все тесты работают корректно.

2. Реализуйте методы `__init__()`, `__str__()`, `add_card()` в классе `Hand` (рука). Рекомендуется использовать список карт для моделирования руки.
   При реализации метода `__str__()` используйте функцию `str()` для получения описания каждой карты (`Card`).

3. Напишите doctest'ы для проверки реализации класса Hand.
   Тестирование можно организовать следующим образом:

   a) создайте несколько объектов класса Card;
   b) создайте объект класса Hand, добавьте в него одну из карт;
   c) напечатайте Hand и проверьте, что печать соответствует добавленной карте;
   d) добавьте ещё карт, распечатайте и проверьте результаты.

4. Реализуйте методы класса `Deck` (колода) согласно имеющемуся шаблону. Для моделирования колоды используйте список карт. Начните с создания всех карт с помощью двух вложенных циклов: один по масти (англ. suit), а другой по категории (англ. rank). Для создания каждой карты используйте конструктор класса `Card`.
   Для перемешивания колоды используйте функцию `random.shuffle()`.

5. Проверьте реализацию класса Deck, используя doctest'ы.
   a) Сделайте тест, проверяющий корректность создания колоды конструктором. Для этого распечатайте колоду сразу после создания и проверьте, что она включает все карты в правильном порядке.
   b) Сделайте тест, проверяющий правильность работы метода deal_card(). Для этого можно несколько раз запрашивать карту из колоды, преобразовывать её в строку и проверять, что перед выдачей текстовое описание колоды начиналось (или

заканчивалось, если карты вынимаются из конца колоды) именно с этих букв, а после выдачи их там больше нет.

c) Проверьте перемешивание колоды. Это не просто, так как перемешивание всегда даёт случайный результат. Но можно проверить три условия: во-первых, длина описания колоды не должна измениться после перемешивания; во-вторых, описание должно стать другим; в третьих карты в колоде не должны повторяться. Для проверки последнего условия достаточно вынимать из колоды все карты по одной и добавлять их в отдельный список, проверяя, что раньше такой карты там не было. Если все проверки сработают нормально, значит все карты уникальны.

6. Реализуйте обработчик для кнопки «Раздать» (англ. deal), который должен перемешать карты и раздать по две карты игроку и дилеру.
   Этот обработчик должен перемешать колоду (её следует хранить как глобальную переменную), создать новые руки для игрока и дилера (их также следует хранить как глобальные переменные), и добавить в каждую руку по две карты.
   Для добавления карты следует использовать метод `deal_card()` класса `Deck` и метод `add_card()` класса `Hand`.
   Полученные руки должны быть распечатаны на консоль с сообщением, показывающим, какая рука принадлежит игроку, а какая – дилеру.

7. Реализуйте метод `get_value()` класса `Hand`. Следует использовать словарь VALUE (имеется в шаблоне) со значениями карт.
   Для расчёта стоимости руки используйте следующий метод. Вначале сложите стоимость всех карт, считая стоимость туза за 1. Во время сложения запомните (используя переменную булевского типа), встречался ли в руке туз. Если туз был, добавьте к сумме 10, если это не приводит к «перебору» (полученная сумма не должна превысить 21).
   Чтобы понять, почему этот метод подсчёта очков работает корректно, обратитесь к разделу «Наблюдения за ходом игры».
   Проверьте реализацию метода `get_value()`, используя шаблон value_template.py.

8. Реализуйте обработчик для кнопки «Взять ещё» (англ. hit). Если значение руки игрока ≤ 21, обработчик должен добавить в неё ещё одну карту. Если значение руки превысит 21, на консоли должно быть напечатано сообщение «Перебор».

9. Реализуйте обработчик для кнопки «Хватит» (англ. stand). Если игрок перебрал, напомните ему об этом. В противном случае в цикле добавляйте карты в руку дилера, пока её стоимость не станет ≥ 17. Используйте цикл `while`.
   Если дилер перебрал, распечатайте соответствующее сообщение. Иначе сравните очки игрока и дилера и распечатайте сообщение о победителе. Помните, что в разрабатываемой версии игры при равенстве очков выигрывает дилер.

В предложенном варианте игра автоматически раздаёт карты игроку и дилеру при запуске. Это реализовано с помощью вызова функции `deal()` во время инициализации.

После реализации этой части программы рекомендуется провести её тщательное и всестороннее тестирование.

Реализация интерфейса

Убедившись, что логика программы реализована правильно, переходите ко второму этапу разработки.

При реализации функций, подразумевающих вывод на экран значений глобальных переменных, не забывайте инициализировать их начальные значения (такие, как создание пустой руки для игрока и дилера) непосредственно перед запуском окна.

1. Реализуйте метод `draw()` класса `Hand`, используя для рисования каждой карты одноимённый метод класса `Card`.
   Рекомендуется рисовать руку как последовательность карт, расположенных в одну строку по горизонтали. Параметр `pos` используйте как координаты верхнего левого угла самой левой карты (рис. 7.7).
   Для упрощения кода считайте, что на холсте должно быть видно не более 5 карт игрока.

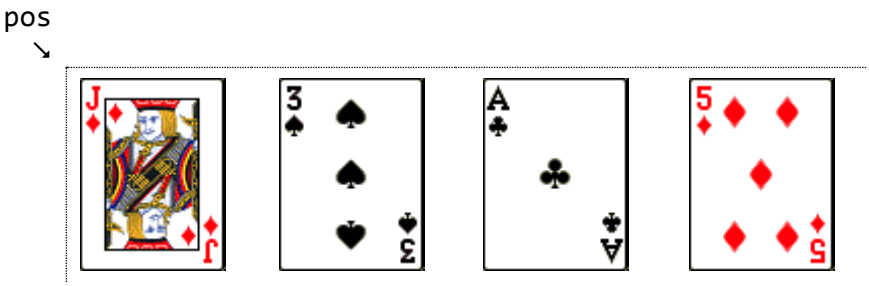


Рис. 7.7. Вывод руки

2. Замените вывод сообщений в консоли отображением сообщения на холсте. Рекомендуется использовать глобальную строковую переменную `outcome` (от англ. исход), которую следует выводить в обработчике рисования `draw()` с помощи функции `draw_text()`. Сообщения должны запрашивать действия у пользователя и иметь форму вопроса, например: «Ещё или хватит?», «Раздать снова?».
   Нарисуйте название игры «Блекджек» на холсте.

3. Добавьте глобальную булеву переменную `in_play`, которая должна отслеживать, продолжается ли ещё ход игрока. Если игрок все ещё продолжает выбирать, следует рисовать рубашку перевёрнутой карты поверх первой карты руки дилера. Когда игрок останавливается, следует прекратить скрывать руку дилера.

4. Добавьте счётчики числа выигрышей и поражений для сессии игры.

5. Измените логику обработчика кнопки «Раздать», чтобы он каждый раз создавал и перемешивал новую колоду. Это позволит избежать ситуации, когда колода становится пустой во время игры.

6. Измените логику обработчика копки «Раздать», чтобы при нажатии на эту кнопку во время игры, программа сообщала о проигрыше игрока и соответственно корректировала очки.

Дополнительные задания

Реализуйте последовательный вывод карт руки дилера, после того, как игрок остановится. Выводите каждую новую карту дилера через секунду. Используйте таймер и глобальную переменную, отслеживающую, сколько карт можно показывать. Максимальное число карт для отображения можно передавать в качестве параметра функции `draw()` класса `Hand`.

Оценка задания

Игра отображает руки игрока и дилера текстовыми сообщениями[2] – 10 %

или

игра отображает руки игрока и дилера в графическом режиме[2] – 20 %.

Первая карта дилера скрыта во время хода игрока – 10 %.

Нажатие на кнопку «Раздать» перемешивает колоду и раздаёт по две карты – 20 %.

Нажатие на кнопку «Раздать» в процессе игры засчитывается как проигрыш – 5 %.

Нажатие на кнопку «Ещё» приводит к выдаче карты игроку – 10 %.

Нажатие на кнопку «Хватит» приводит к выдаче карт дилеру по правилам игры – 10 %.

Программа распознаёт перебор игрока и дилера – 10 %.

Программа корректно определяет и объявляет победителя – 5 %.

Программа корректно выводит приглашения вида «Ещё или хватит?», «Раздать снова?» – 5 %.

Программа корректно ведёт счёт выигрышей и проигрышей – 5 %.

Дополнительные баллы

Последовательный вывод карт дилера в конце раунда – 15 %.

Срок сдачи задания.

Для подгруппы, занимающейся по вторникам: 8 ноября 2016;

для подгруппы, занимающейся по пятницам: 1 ноября 2016.

В случае сдачи задания с опозданием, полученные за него баллы будут уменьшены:

- при задержке на 1 неделю: баллы умножаются на 0.9;
- при задержке на 2 недели: баллы умножаются на 0.75;
- при задержке на 3 и более недель: баллы умножаются на 0.65.

[2] Баллы за первые два пункта начисляются только при наличии doctest'ов.