9. Астероиды

Постановка задачи

В этом домашнем задании будет продолжена работа над игрой «Астероиды».

В ходе решения задания игра будет дополнена возможностями работы с несколькими астероидами и ракетами одновременно. Игрок будет терять «жизни» при столкновении корабля с астероидом и набирать очки, сбивая астероиды ракетами. Потребуется вести подсчёт жизней и завершать игру, когда они закончатся. В качестве дополнительного задания можно будет реализовать анимацию взрывов.

Код шаблона для этого домашнего задания содержит некоторые дополнения по сравнению с предыдущим шаблоном. Скопируйте в него реализацию классов `Ship` и `Sprite`, а также обработчики клавиатуры и таймера из предыдущего задания.

Рекомендованный порядок работы над заданием

Процесс доработки игры можно разделить на пять этапов.

Этап 1. Несколько астероидов

Вначале реализуйте поддержку работы с несколькими астероидами.

Для этого действуйте в следующем порядке:

1. Удалите переменную `a_rock`, а вместо неё введите переменную для множества астероидов `rock_group`.
   Инициализируйте `rock_group` пустым множеством.
   Модифицируйте код создания астероидов (обработчик таймера) так, чтобы он каждый раз создавал новые астероиды (объекты класса `Sprite`) и добавлял их в `rock_group`.

2. Модифицируйте код создания астероидов так, чтобы он не создавал слишком много астероидов. Рекомендуется ограничить число одновременно используемых астероидов двенадцатью. При большем числе астероидов игра становится менее интересной.
   Для этого проверяйте, сколько астероидов уже есть в `rock_group`, и создавайте новый, только если лимит ещё не набран.

3. Создайте вспомогательную функцию `process_sprite_group()`. Аргументами этой функции должны быть множество спрайтов и холст. Для каждого спрайта функция должна вызвать методы `update()` и `paint()`.

4. Вызовите функцию `process_sprite_group()` из обработчика рисования. Эта функция должна обрабатывать множество астероидов `rock_group`.

В конце первого этапа в игре каждую секунду должен создаваться новый астероид. При достижении заданного лимита появление новых астероидов должно прекратиться. Все астероиды должны перемещаться независимо друг от друга.

Этап 2. Столкновения

Теперь добавьте столкновения между кораблём и астероидами.

1. Добавьте метод `collide()` (англ. столкновение) к классу `Sprite`.
   Аргументом метода должен быть `other_object` – «другой объект», столкновение с которым проверяется.
   Метод должен возвращать значение `True`, если зафиксировано столкновение, и `False` – если нет.
   Пока в качестве «другого объекта» будет выступать корабль, но далее эта же функция будет использована для проверки столкновений с ракетами.
   Для определения столкновения используйте координаты центров и радиусы объектов. Для получения этих данных добавьте в классы `Ship` и `Sprite` пары методов `get_position()` и `get_radius()`.

2. Реализуйте вспомогательную функцию `group_collide()`.
   Аргументами этой функции должны быть множество спрайтов group и спрайт `other_object`. Функция должна для всех спрайтов группы проверить столкновение с `other_object`.
   Для проверки столкновения используйте метод `collide()` класса `Sprite`.
   Если зафиксировано столкновение, то столкнувшийся объект должен быть удалён из группы. Не забудьте, что внутри цикла `for … in …` нельзя модифицировать структуру, по которой проводится итерация. Используйте один из вариантов удаления, рассмотренный в примере balls-del.py.
   Функция должна возвращать значение `True`, если зафиксировано хотя бы одно столкновение, и `False` – если нет.

3. В обработчике рисования используйте функцию `group_collide()` для проверки столкновения корабля с астероидами. В случае столкновения уменьшайте число жизней на 1 (переменная `lives`). Пока допустима игра с отрицательным числом жизней.

На этом этапе игрок может спасаться от астероидов, только улетая от них.

Этап 3. Ракеты

Добавьте поддержку работы с группой ракет.

1. Удалите переменную `a_missile` и введите вместо неё переменную `missile_group` для множества ракет.
   Инициализируйте `missile_group` пустым множеством.
   Модифицируйте метод выстрела `Shoot()` в классе `Ship` так, чтобы при каждом выстреле создавалась новая ракета (объект класса `Sprite`) и добавлялась к множеству `missile_set`.

2. В обработчик рисования добавьте вызов функции `process_sprite_group()` для обработки `missile_group`.
   Теперь при каждом выстреле игрока возникает новая ракета. Но ракеты остаются на всё время игры. Срок их жизни надо ограничить. Для этого потребуется сделать модификации в классе `Sprite` и в функции `process_sprite_group()`.

3. В методе `update()` класса Sprite увеличивайте на 1 возраст спрайта (переменная `self.age`).
   Если возраст станет больше, чем допустимое время жизни (переменная `self.lifespan`), то такой объект следует удалить. Чтобы это можно было сделать, метод `update()` должен

возвращать значение `False` (что будет означать «удалять пока не надо»), если возраст меньше предельного, и значение `True` (объект требует удаления), если достигнуто предельное время жизни.

4. Добавьте проверку значения, возвращаемого методом `update()` в функцию `process_sprite_group()`. Если метод `update()` вернул значение `True`, удалите спрайт из группы.
   Не забудьте о запрете удаления элементов из множества, по которому проходит итерация.

По завершении третьего этапа игрок получил возможность стрелять, однако ракеты пока не сбивают астероиды.

Этап 4. Доработка столкновений

Теперь нужно обеспечить проверку столкновений между ракетами и астероидами. Использовать непосредственно функцию `group_collide()` не получится – необходимо проверять столкновения между двумя группами. Для этого лучше разработать ещё одну вспомогательную функцию.

1. Реализуйте вспомогательную функцию `group_group_collide()`, аргументами которой являются два множества спрайтов.
   В функции реализуйте цикл по копии первого множества. Для каждого спрайта вызывайте `group_collide()` для проверки столкновения со всеми спрайтами второго множества.
   Удаляйте спрайты, для которых зафиксировано столкновение, из первой группы. Для удаления удобно использовать метод `discard()` множества.
   Функция должна вернуть число элементов из первой группы, которые столкнулись с элементами из второй группы.

2. Для проверки столкновений между ракетами и астероидами добавьте вызов функции `group_group_collide()` в обработчик рисования.
   Увеличивайте очки (переменная `score`) за каждый сбитый астероид.

На этом этапе в игру уже можно играть, но пока нельзя проиграть.

Этап 5. Финальная доработка

Добавьте последние штрихи, чтобы завершить игру.

1. Добавьте код в обработчик рисования, чтобы игра перезапускалась, когда число жизней дойдёт до нуля. Для этого установите значение переменной `started` в `False` и прекратите создание новых астероидов и управление кораблём, пока игра остановлена.

2. Убедитесь, что при перезапуске игры счётчики жизней и очков корректно инициализируются.
   При запуске игры начинайте процесс создания астероидов.
   Начинайте воспроизведение музыки в начале игры и перематывайте её при перезапуске.

3. При создании новых астероидов необходимо проверять, чтобы они не создавались слишком близко к кораблю. Иначе корабль может погибнуть от столкновения с возникшим прямо на нём астероидом, что не честно по отношению к игроку.
   Самый простой способ решить эту проблему – игнорировать создание астероида, если он находится слишком близко к кораблю (не добавляйте его в `rock_group`).

4. Попробуйте изменять начальную скорость астероидов пропорционально текущему счёту, чтобы игра усложнялась по мере набора очков.

5. Поэкспериментируйте со значениями разных констант, чтобы сделать игру интересной.

Дополнительные задания

В качестве расширения игры реализуйте анимацию взрывов.

Сохраните копию своего кода перед тем, как попробовать реализовать это расширение.

1. В методе `draw()` класса `Sprite` проверьте, имеет ли переменная `self.Animated` значение `True`.
   Для анимированных спрайтов выберите соответствующий тайл, используя возраст (переменная `self.age`) в качестве индекса, и выводите этот тайл.
   Если спрайт не является анимированным, рисуйте его как прежде.

2. Создайте множество `exploision_group`, инициализируйте его пустым множеством.

3. При обнаружении столкновения в методе `group_collide` создайте новый взрыв (объект класса `Sprite`) и помещайте его в `exploision_group`.
   При каждом взрыве воспроизводите звук взрыва.

4. В обработчике рисования добавьте вызов функции `process_sprite_group()` для обработки взрывов.

Оценка задания

Игра создаёт и рисует несколько астероидов	– 10 %.
Игра проверяет столкновения корабля и астероидов	– 10 %.
После столкновения с кораблём астероид исчезает	– 10 %.
После столкновения корабля с астероидом игрок теряет одну жизнь	– 5 %.
Игра работает с несколькими ракетами	– 10 %.
Ракеты, которые не попали в астероид, исчезают через некоторое время	– 10 %.
Игра проверяет столкновения между астероидами и ракетами	– 10 %.
После столкновения ракеты и астероида оба пропадают	– 10 %.
Игрок получает очки за сбитые астероиды	– 10 %.
Игра завершается и появляется заставка, если число жизней достигает 0	– 5 %.
При щелчке на заставке число жизней устанавливается в 3, очки – в 0, музыка перезапускается	– 5 %.

В режиме заставки игра не создаёт новых астероидов и не управляет кораблём – 5 %.

Дополнительные баллы

Анимация взрывов – 10 %.

Звук взрывов – 5 %.

Срок сдачи задания.

Для подгруппы, занимающейся по вторникам: 29 ноября 2016;

для подгруппы, занимающейся по пятницам: 2 декабря 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.