**Mini-project development process**

At the end of this project, your game will have multiple rocks and multiple missiles.  You will lose a life if your ship collides with a rock and you will score points if your missile collides with a rock. You will keep track of the score and lives remaining and end the game at the proper time.  You may optionally add animated explosions when there is a collision.

**Phase one -  Multiple rocks**

For this phase, you will keep a set of rocks and spawn new rocks into this set.  This requires the following steps:

1. Remove a\_rock and replace it with rock\_group. Initialize the rock group to an empty set. Modify your rock spawner to create a new rock (an instance of a Sprite object) and add it to rock\_group.
2. Modify your rock spawner to limit the total number of rocks in the game at any one time. We suggest you limit it to 12. With too many rocks the game becomes less fun and the animation slows down significantly.
3. Create a helper function process\_sprite\_group. This function should take a set and a canvas and call the update and draw methods for each sprite in the group.
4. Call the process\_sprite\_group function on rock\_group in the draw handler.

**Phase two - Collisions**

For this phase, you will detect collisions between the ship and a rock. Upon a collision, the rock should be destroyed and the player should lose a life. To implement ship-rock collisions, you need to do the following:

1. Add a collide method to the Sprite class. This should take an other\_object as an argument and return True if there is a collision or Falseotherwise. For now, this other object will always be your ship, but we want to be able to use this collide method to detect collisions with missiles later, as well. Collisions can be detected using the radius of the two objects. This requires you to implement methods get\_position andget\_radius on both the Sprite and Ship classes.
2. Implement a group\_collide helper function. This function should take a set group and a sprite other\_object and check for collisions betweenother\_object and elements of the group. If there is a collision, the colliding object should be removed from the group. To avoid removing an object from a set that you are iterating over (which can cause you a serious debugging headache), iterate over a copy of the set created viaset(group). This function should return True or False depending on whether there was a collision. Be sure to use the collide method from part 1 on the sprites in the group to accomplish this task.
3. In the draw handler, use the group\_collide helper to determine if the ship hit any of the rocks. If so, decrease the number of lives by one. Note that you could have negative lives at this point. Don't worry about that yet.

At this point, you should have a game of "dodge 'em". You can fly around trying to avoid the rocks!

**Phase three - Missiles**

For this phase, you will keep a set of missiles and spawn new missiles into this set when firing using the space bar. This requires the following steps:

1. Remove a\_missile and replace it with missile\_group.  Initialize the missile group to an empty set.  Modify your shoot method of my\_ship to create a new missile (an instance of the Sprite class) and add it to the missile\_group. If you use our code, the firing sound should play automatically each time a missile is spawned.
2. In the draw handler, use your helper function process\_sprite\_group to process missile\_group. While you can now shoot multiple missiles, you will notice that they stick around forever. To fix this, we need to modify the Sprite class and the process\_sprite\_group function.
3. In the update method of the Sprite class, increment the age of the sprite every time update is called. If the age is greater than or equal to the lifespan of the sprite, then we want to remove it. So, return False (meaning we want to keep it) if the age is less than the lifespan and True(meaning we want to remove it) otherwise.
4. Modify process\_sprite\_group to check the return value of update for sprites. If it returns True, remove the sprite from the group. Again, you will want to iterate over a copy of the sprite group in process\_sprite\_group to avoid deleting from the same set over which you are iterating.

**Phase four - Collisions revisited**Now, we want to destroy rocks when they are hit by a missile. We can't quite use group\_collide, because we want to check for collisions between two groups. All we need to do is add one more helper function:

1. Implement a final helper function group\_group\_collide that takes two groups of objects as input. group\_group\_collide should iterate through the elements of a copy of the first group using a for-loop and then call group\_collide with each of these elements on the second group.group\_group\_collide should return the number of elements in the first group that collide with the second group as well as delete these elements in the first group. You may find the discard method for sets to be helpful here.
2. Call group\_group\_collide in the draw handler to detect missile/rock collisions. Increment the score by the number of missile collisions.

**Phase five - Finish it off**

You now have a mostly working version of *RiceRocks*!!! Let's add a few final touches.

1. Add code to the draw handler such that, if the number of lives becomes 0, the game is reset and the splash screen appears. In particular, set the flag started to False, destroy all rocks and prevent any more rocks for spawning until the game is restarted.
2. When the game starts/restarts, make sure the lives and the score are properly initialized. Start spawning rocks. **Play/restart the background music** loaded in the variable soundtrack in the program template.
3. When you spawn rocks, you want to make sure they are some distance away from the ship. Otherwise, you can die when a rock spawns on top of you, which isn't much fun. One simple way to achieve this effect to ignore a rock spawn event if the spawned rock is too close to the ship.
4. Experiment with varying the velocity of rocks based on the score to make game play more difficult as the game progresses.
5. Tweak any constants that you have to make the game play the way you want.

Congratulations! You have completed the assignment. Enjoy playing your game!!!

**Bonus**

The following will not be graded. Feel free to try this, but do not break any of the other game functionality. We strongly recommend that you save your work before doing this and keep track of it, so you can submit a working version of the first five phases if you end up breaking your game trying to add more features.

One thing that is missing in your game is explosions when things collide. We have provided a tiled explosion image that you can use to create animated explosions. To get things working, you will need to do a few things:

1. In the draw method of the Sprite class, check if self.animated is True. If so, then choose the correct tile in the image based on the age. The image is tiled horizontally. If self.animated is False, it should continue to draw the sprite as before.
2. Create an explosion\_group global variable and initialize it to an empty set.
3. In group\_collide, if there is a collision, create a new explosion (an instance of the Sprite class)and add it to the explosion\_group. Make sure that each explosion plays the explosion sound.
4. In the draw handler, use process\_sprite\_group to process explosion\_group.

You should now have explosions working!

For more helpful tips on implementing this mini-project, please visit the [Code Clinic tips](https://class.coursera.org/interactivepython2-003/wiki/ricerocks_tips) page for this mini-project.

**Grading rubric - 13 pts (scaled to 100 pts)**

Note that the animated explosions are not graded. However, please add comments concerning the quality of the explosions and general gameplay in the free comments at the bottom of the page. Please assess your peer's mini-projects in Chrome. **If, for some reason, you must use Firefox or another browser (or had issues playing sounds in Chrome), please give your peers full credit on the sound-related rubric items.**

* 1 pt - The program spawns multiple rocks.
* 1 pt - The program correctly determines whether the ship collides with a rock.
* 1 pt - The program removes a rock when the ship collides with a rock.
* 1 pt - The number of lives decreases by one when the ship collides with a rock.
* 1 pt - The program spawns multiple missiles.
* 1 pt - The program plays the firing sound when each missile is spawned.
* 1 pt - The program removes a missile that does not collide with a rock after some fixed time period.
* 1 pt - The program correctly determines whether a missile and a rock collide.
* 1 pt - The program removes missiles and rocks that collide.
* 1 pt - The score is updated appropriately after missile/rock collisions.
* 1 pt - When the lives go to zero, the splash screen reappears and all rocks are removed.
* 1 pt - When the splash screen is clicked, the lives are reset to 3, score is reset to zero and the background music restarts.
* 1 pt - The game spawns rocks only when the splash screen is not visible and a game is in progress.