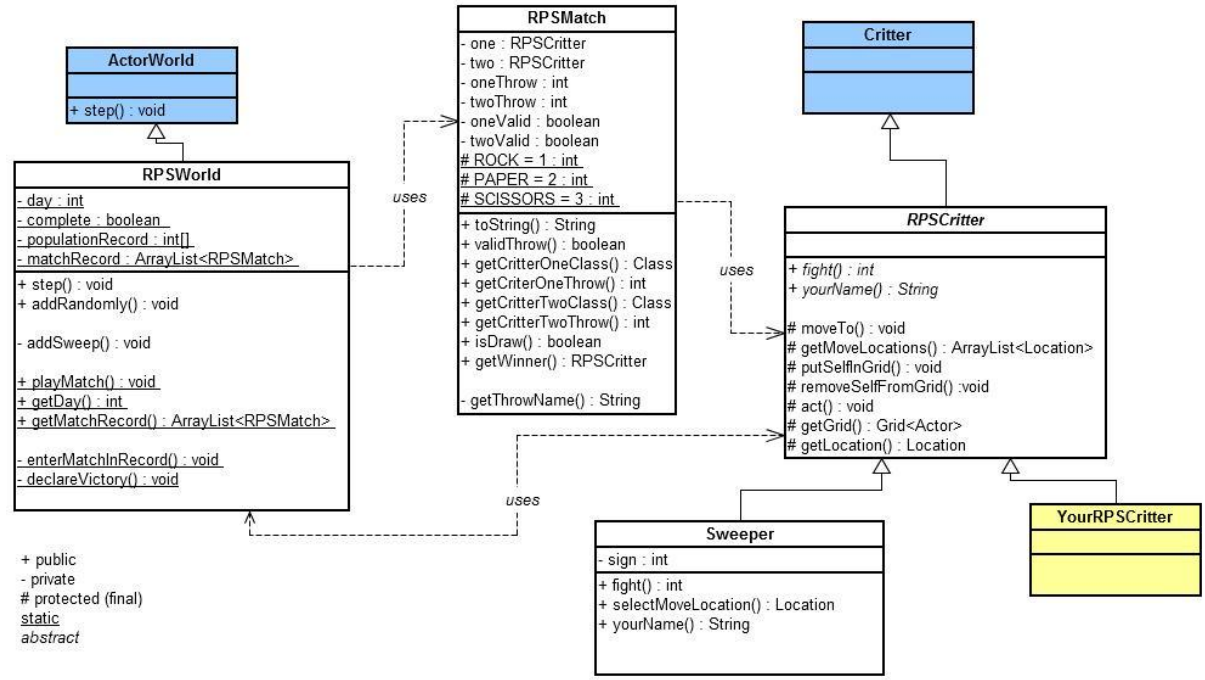
Rock-Paper-Scissors Critter Lab

Try your hand (no pun intended) against the New York Times interactive Rock-Paper-Scissors AI. While random play would always result in random winners (with the overall score expected to be even in the long run), knowing an opponent's tendencies allows you to develop a strategy.

You'll need a regular GridWorld installation plus the RPScritters.zip (download from Moodle) that includes RPSWorld, RPSMatch, RPSRunner, the RPSCritter base class and some example Critters. Be sure to copy the .gif image files to the same place where your class files are output (probably the same directory as your JCreator workspace / project file). Note carefully the restrictions on subclasses of RPSCritter! Also note the static methods that RPSWorld offers you. You will need to add code to the abstract method fight(), and you may override the inherited method selectMoveLocation() (Note: You may not override getMoveLocations(), but can look at the locations surrounding your choices in order to make the best move. See Hint#1 below for more details.)

RPS Class Diagram



The Rules of RPSWorld

1. Neighbors always fight
2. Moving to someone else's Location causes a fight.
3. Fights are settled by Rock-Paper-Scissors
4. Every RPSCritter is a Critter

At a minimum, your RPSCritter shouldn't win or lose to members of its own class. In other words, don't kill your own kind! It should also be fairly easy for you to win against Critters that always throw the same sign, like the PaperCritter.

Hints

1. The "selectMoveLocation" strategy. It's important to note that getMoveLocations is marked final in RPSCritter, and so following the intended design of Critter means that RPSCritter subclasses move (at most) one space per step. Another way to explain this is to say, "selectMoveLocation is not allowed to construct new Locations!" Instead, selectMoveLocation must return one of the values from the ArrayList locs that was passed in from getMoveLocations. The movement strategy is important because avoiding fights is the best way not to lose. Note that getActors and processActors can be overridden to catalog the number and type and position of other Actors on the grid in order to set an instance variable (maybe "preferredDirection"?) that can be later used by selectMoveLocation.

2. The "fight" strategy. The minimum fight strategy should insure that an RPSCritter doesn't lose to another instance of the same subclass. That means you will need to use *instanceof* on the passed-in opponent. A decent fight strategy should also not lose to opponent classes that do the same throw every time. The included RockCritter, PaperCritter, and ScissorsCritter are good examples of classes that you don't want to lose to. You should try and dynamically determine your opponent's habits. This is where getMatchRecord becomes useful. You can loop over that ArrayList in order to identify their opponent in the RPSMatch.

In your `fight` method, this is a pretty useful line of code:

```
ArrayList<RPSMatch> matchRecord = RPSWorld.getMatchRecord(opponent.getClass());
```

Then you can go through the matchRecord array and figure out whatever statistics you think are important. Beware, `RPSWorld.getMatchRecord` will return null if the opponent class has not yet played any matches!

Submitting your RPSCritter

In the comment section of your RPSCritter, be sure to include:

1. Your name and period.
2. A description of the movement strategy you used
3. A description of the throw (fight) strategy you used

Submit your RPSCritter .java file and .gif file to Moodle. Check our calendar for due dates and competition dates.