Assignment 2: Step

Monster High

Introduction

In this assignment, we'll write Step code to select and execute content units. Our content units are plot points for a soap opera world called *Monster High*. In this world, students, who also happen to be monsters, engage in conflicts and romance.

The provided code sets up the framework for the assignment. You'll be adding code to two of the files, `Queries.step` and `PlotPoints.step`, as well as potentially changing the initial state of the world in `Students.step` to test your code.

This code works by testing the state of the world to see which plot points are potentially applicable and then picking one to execute. Two entry points have been provided for testing the code. The task `[Events]` selects and executes six random plot points. The task `[ShowPlotPointMenu]` shows a menu of all the plot points available in the current story state. The user can select one, it executes, and a menu of plot points available in the new state is displayed. Plot points will be represented as tuples, so if you haven't done the reading on tuples, do that now.

For example, we might represent the plot point that Jayden confesses their love to Tiana with the tuple: `[confess_love jayden tiana]`. However, this only makes sense if we know that Jayden has a crush on Tiana, otherwise, why would they confess? And it doesn't make sense if the two are already dating. So you need to make sure that the `confess_love` plot point is only available when there's an attraction that isn't being acted on. When this plot point executes, it might be printed `as "Jayden confesses their love to Tiana"`. Additionally, when the plot point executes, it changes the state of the world so that Tiana now knows that Jayden as a crush on Tiana: `Knows Tiana [CrushOn Jayden Tiana]`. Here we're also using tuples to represent what it is Tiana knows.

We're using `fluents` to represent the changing state of the world as plot points execute. Read more about fluents in tutorial 8.

Note also that we've provided you with a version of `Mention` that understands things like pronoun generation and capitalization of character names. You don't need to do anything with it, just realize that it's there. If you don't know what `Mention` is, read part 6 of the tutorial on generating text in context.

Getting started

To edit Step code, it is recommended you use Microsoft Visual Studio Code as there is a syntax highlighting extension available in VS Code for Step. If you decide to use VS Code, follow the directions under *Using Visual Studio Code* in the *StepRepl Guide* documentation.

If you're using the Mac version of StepREPL, you will likely see a message saying the app can't be run. This is the Mac security policy responding to a downloaded unsigned app and marking it as quarantined. Open the unix terminal and execute the command:

```
xattr -rd com.apple.quarantine StepRepl
```

You should now be able to run the app.

In StepREPL, use “Open Project…” in the File menu to open the folder “Monster High Assignment Winter 2025”

Optional: Making a student body

The file `Students.step` defines the student body, including what type of monster each student is, what clubs people are in, and who has crushes, is dating, and are friends. For the purposes of testing your code, you may want to create your own student body with a specific initial state. However, you don’t need to do this if you don’t want to.

Note that the `Mention` code we gave you will use *they* as the pronoun for all characters by default. If you prefer, you may optionally specify preferred pronouns for some or all of your characters. To do that, just add statements to your `Students.step` file of the form:

> `PreferredPronoun` *student pronoun*`.`

Where student is the *student* you’re specifying a pronoun for, and *pronoun* is either `he` or `she`. You can specify `they`, but since that’s the default anyway, there’s no particular point in doing so. Anyone you don’t specify a pronoun for will be referred to using `they`.

Part one: Queries

The file `Queries.Step` defines predicate rules to infer interesting relations in the story world. Examples include:
`UnrequitedLove ?a ?b` (?a is crushing on ?b, but not vice versa) and,
`CheatingOn ?cheater ?cheatee` (?cheater is dating ?cheatee and is also dating someone else).
Write predicate rules for each of the predicates specified in `Queries.Step`.

Part two: Plot Points

Here’s how plot point selection works. Calling `[PlotPoint ?event]` picks plot points at random to test and returns a binding for `?event` for the first plot point it finds whose test is satisfied. For example, if the `confess_love` plot point between jayden and triana was the first plot point it tried whose test was satisfied, then `[PlotPoint ?event]` succeeds with `?event = [confess_love jayden triana]`.

`PlotPoint` is a predicate: `PlotPoint ?event` means that `?event` is a plausible thing to happen in a Monster High episode given the characters and relationships specified in `Students.step`.

If you wanted to see all the plot points that are possible in the current story state, you can use: `[FindUnique ?event [PlotPoint ?event] ?eventList]`. `?eventList` will be bound to a list of all the possible plot points.

We've defined two hot keys, Alt-e/Option-e, which executes six plot points in a row, and Alt-m/Option-m, which shows a menu of all the possible plot points, allowing you to pick one to execute (you then see a new menu).

For each plot point, you need to write a method for the `PlotPoint` task to test if a plot point is available in the current story state.

We represent plot points as tuples:

- `[confess_love ?a ?b]`
  means `?a` confesses their love for `?b`. This can occur when `?a` has a crush on `?b` but they aren't already dating.
- `[start_dating ?a ?b]`
  means `?a` starts dating `?b`. There are two different conditions in which `?a` can start dating `?b`:
  - `?a` knows that `?b` has a crush on them, there's a mutual attraction, they are not currently dating each other, and they are not star crossed lovers.
  - Same as the above, except that they are star crossed lovers, and `?a` is confident.
- `[romantic_rejection ?rejector ?rejected]`
  means `?rejector` romantically rejects `?rejected`. This can occur when the `?rejector` knows that `?rejected` has a crush on them, and the love is unrequited.
- `[fight ?attacker ?defender [triangle ?attacker ?defender ?loveInterest]]`
  means `?attacker` has a fight with `?defender` about the fact that `?attacker` and `?defender` have the same `?loveInterest`. This can occur when `?attacker` and `?defender` are in a love triangle with `?loveInterest`.
- `[fight ?cheatee ?cheater [cheating ?cheater ?cheatee ?other]]`
  means that `?cheatee` fights with `?cheater` about `?cheater` cheating on them with other. This can occur when `?cheater` is cheating on `?cheatee` with `?other`.
- `[fight ?cheatee ?other [cheating ?cheater ?cheatee ?other]]`
  means that `?cheatee` fights with `?other` about `?cheater` cheating on `?cheatee` with `?other`. This can occur when `?cheater` is cheating on `?cheatee` with `?other`.
- `[breakup ?char1 ?char2]`
  means that `?char1` breaks up with `?char2`. This can occur when `?char1` and `?char2` are dating and `?char1` had a fight with `?char2`.
- `[smoldering_look ?a ?b ?club]`
  means `?a` gave ?b a smoldering look when they were together at `?club`. This can occur when the characters are mutually attracted to each other and they both belong to the club.

- `[star_crossed_lovers ?a ?atype ?b ?btype]`
  can only occur when `?a` and `?b` are star crossed lovers (see definition in `Queries.step`).
- `[support_from_friend ?supporter ?supportee [star_crossed_lovers ?supportee ?supporteeType ?crush ?crushType]]`
  means that `?supporter` supports `?supportee` about `?supportee` being star crossed lovers with `?crush`. This can only occur if `?supportee` is star crossed lovers with `?crush` and `?supporter` and `?supportee` are friends.

So for this part, you need to add methods for `PlotPoint` to the end of `PlotPoints.step`. The methods are going to look like:

`PlotPoint [`*eventType other-information …*`]:` *condition*

Where:

- *eventType* is the kind of event (e.g. `confess_love`, `star_crossed_lovers`, etc.)
- *other-information* is the other information that appears in that kind of event: definitely the characters participating in it, but for some of them, there are things like the monster types or clubs the students belong to
- *condition* is the information to determine whether this plot point is allowed in the current state of the story world.

Here's an example. For the breakup plot point, it only makes sense for it to happen if the characters are already dating and have had a fight. So the rule for that would look like:

```
PlotPoint [breakup ?a ?b]: [Dating ?a ?b] [HadFightWith ?a ?b]
```

This says that `[breakup ?a ?b]` is a valid plot point provided `[Dating ?a ?b]` and that they had previously had a fight `[HadFightWith ?a ?b]`.

Add methods at the end of `PlotPoints.step` for each of the types of plot points and make sure they only succeed in plausible circumstances.

Testing plot points

You can test plot points in a few different ways:

- Running: `PlotPoint ?`
  That is, typing `PlotPoint ?` in the green box, and hitting return, will generate a completely random plot point and print it in tuple form.
- Running: `PlotPoint [`*type args …*`]`
  Will force it to try to generate a plot point of the specified type. For example:
  - `PlotPoint [confess_love ?a ?b]`
    Will find a character ?a for which they meet the conditions to confess their love to ?b.

- `PlotPoint [confess_love cameron hailey]`
  Will test whether `[confess_love cameron hailey]` is a valid plot point. If it doesn't print anything, it's valid. If it prints Call Failed, then that's not a valid plot point. This isn't a valid plot point in the `Students.step` file we hand out because Cameron and Hailey are already dating. But it might be true in your story world.

Part three: Executing Plot Points

Now write methods for `ExecutePlotPoint`. These should take the form of:

`ExecutePlotPoint [`*eventType other-information …*`]:` *text-to-print state-to-change*

Where, *eventType* and *other-information* are as above, *text-to-print* is what to print to describe this kind of plot point, and *state-to-change* is code to make state changes caused by the plot point.

Here are the state changes the need to happen for each plot point.

- `[confess_love ?a ?b]`
  Now `?b` knows that `?a` has a crush on `?b`.
- `[start_dating ?a ?b]`
  Now `?a` and `?b` are dating, and `?a` doesn't know anymore about `?b`'s crush on `?a` (now that they're dating, they are beyond worrying about crushes).
- `[romantic_rejection ?rejector ?rejected]`
  Now `?rejector` doesn't know anymore that `?rejected` has a crush on them, `?rejected` no longer has a crush on `?rejector`, and `?rejected` considers `?rejector` an enemy.
- `[fight ?attacker ?defender [triangle ?attacker ?defender ? loveInterest]]`
- `[fight ?cheatee ?cheater [cheating ?cheater ?cheatee ?other]]`
- `[fight ?cheatee ?other [cheating ?cheater ?cheatee ?other]]`
  In all three case of fight, now the two characters involved have had a fight. The text will be different for three cases.
- `[breakup ?char1 ?char2]`
  Since the two characters aren't dating anymore, now they haven't had a fight (the fight is no longer relevant after the breakup), and neither has a crush on the other.
- `[smoldering_look ?a ?b ?club]`
  Only prints text. No state changes.
- `[star_crossed_lovers ?a ?atype ?b ?btype]`
  Only prints text. No stage changes.
- `[support_from_friend ?supporter ?supportee [star_crossed_lovers ? supportee ?supporteeType ?crush ?crushType]]`
  Now the `?supportee` is confident.

The text to print and the state changes will need to refer to the variables in the plot point. Here's an example for the confess_love plot point:

```
ExecutePlotPoint [confess_love ?confessor ?confessee]:
      ?confessor confesses ?confessor/Possessive love to ?confessee.
      [now [Knows ?confessee [CrushOn ?confessor ?confessee]]]
[end]
```

One way to test `ExecutePlotPoint` is by first visualizing the story world with the `Visualize` menu command. Then select a plot point from the menu of currently available plot points (using the `Show plot point menu` command) and visualize the story world again to verify the state changed correctly. You can also check if the state change correctly changed the next available plot points.

There are methods defined in `Mention.step` to help with writing text for `ExecutePlotPoint`. The subsections below describe these.

Pronoun generation

As discussed above, if text uses a character's name twice, it will generate pronouns to refer to them when appropriate. By default, it uses they as everyone's pronoun, but you can specify pronouns for specific character using `PreferredPronoun` (see above). That's purely up to you, though.

What you should pay attention to is whether it's generating pronouns in the right case. If your code to print says something like:

```
I love ?who.  ?who is just so great.
```

This will generate something like:

```
I like Jayden.  They are so great.
```

Because the second use of `?who` is the subject of a sentence, this is fine. We want it to say `they` rather than `them`. But if our code says:

```
I love ?who.  I could just hug ?who.
```

Then we get something a little odd:

```
I like Jayden.  I could just hug they.
```

Here we want to use `them` rather than `they` because the second use of `?who` is the object of the verb rather than the subject. This is easy to fix. Just change `?x` to `?x/Obj`:

```
I love ?who.  I could just hug ?who/Obj.
```

This will produce the correct text:

```
I like Jayden.  I could just hug them.
```

When you see your code is generating pronouns in the wrong case, just change it as follows:

- `?x`
  Prints they/she/he

- `?x/Obj`
  Prints them/her/him
- `?x/Possessive`
  Prints their/her/his

Verb conjugation

Hopefully this won't affect you for this assignment. I didn't have to use it for my reference solution, but you may want to generate more ambitious text than I did. As discussed in the reading, you sometimes want to generate different forms of verbs, depending on how the subject is in third person plural or not. We've provided you with the `[Is]` task from the reading, which prints "is" or "are", depending on what's appropriate, as well as `[Has]`, which prints "has" or "have" as appropriate. For example:

```
?a [Is] going out with ?b
```

Will generate "bill is going out with jenny" or "he is going out with jenny", but "they are going out with jenny" when `?who` is realized with the pronoun `they`.

For regular verbs, you can add `[s]` at the end of the verb and it will add an -s or an -es, as appropriate. For example:

```
TestCongation ?who: I like ?who.  ?who read[s] books.
```

Will generate:

```
I like Jayden.  They read books.
```

If Jayden's preferred pronoun is they, but:

```
I like Jayden.  He reads books.
```

If Jayden's preferred pronoun is he.

The magic `[s]` task only works for regular verbs. If you want to handle a different verb, then grab the code for `Is` from `Mention.step` and just change it for your verb.

Part four: Invent some more plot points

Now write `PlotPoint` and `ExecutePlotPoint` methods for two more kinds of plot points. They can be anything of your choosing, although they need to depend on the story world. Feel free to add new information to the story world if you like.

Some ideas:

- Two characters have a heart-to-heart discussion while at their club. This would obviously only make sense when they're in the same club.
- Two characters become friends after a fight. For that, there'd need to be a reason for them to have the fight. And they probably shouldn't be dating, since people who are dating should probably already be friends. Maybe they become friends because they have a shared friend.

- A werewolf character does something on the full moon
- A vampire character bites a human character

Turning it in

Save all your files, reload (control-R) to make sure everything really does load. Assuming everything seems okay, make a zip file of your Monster High directory and upload it to canvas.