Team #12

Members:

Iteration 1: Report

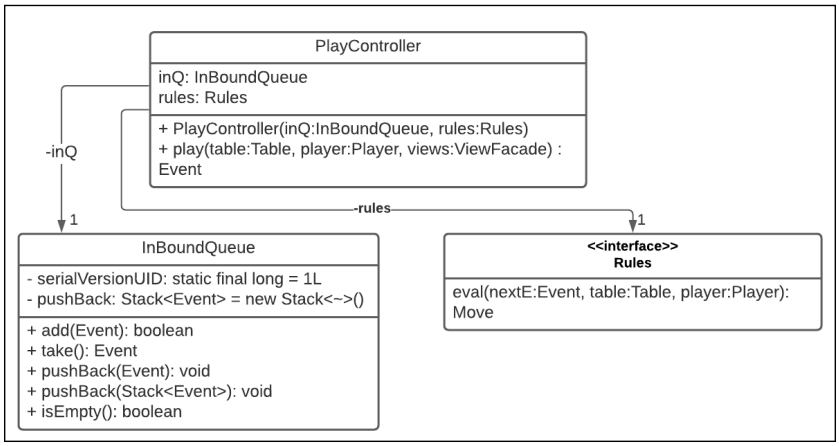
04/01/2021

Members:

- Alexis Cordts
- Andrew Marek
- Evan Christensen
- Jared Weiland
- Steven Sheets
- Zhifan Huang

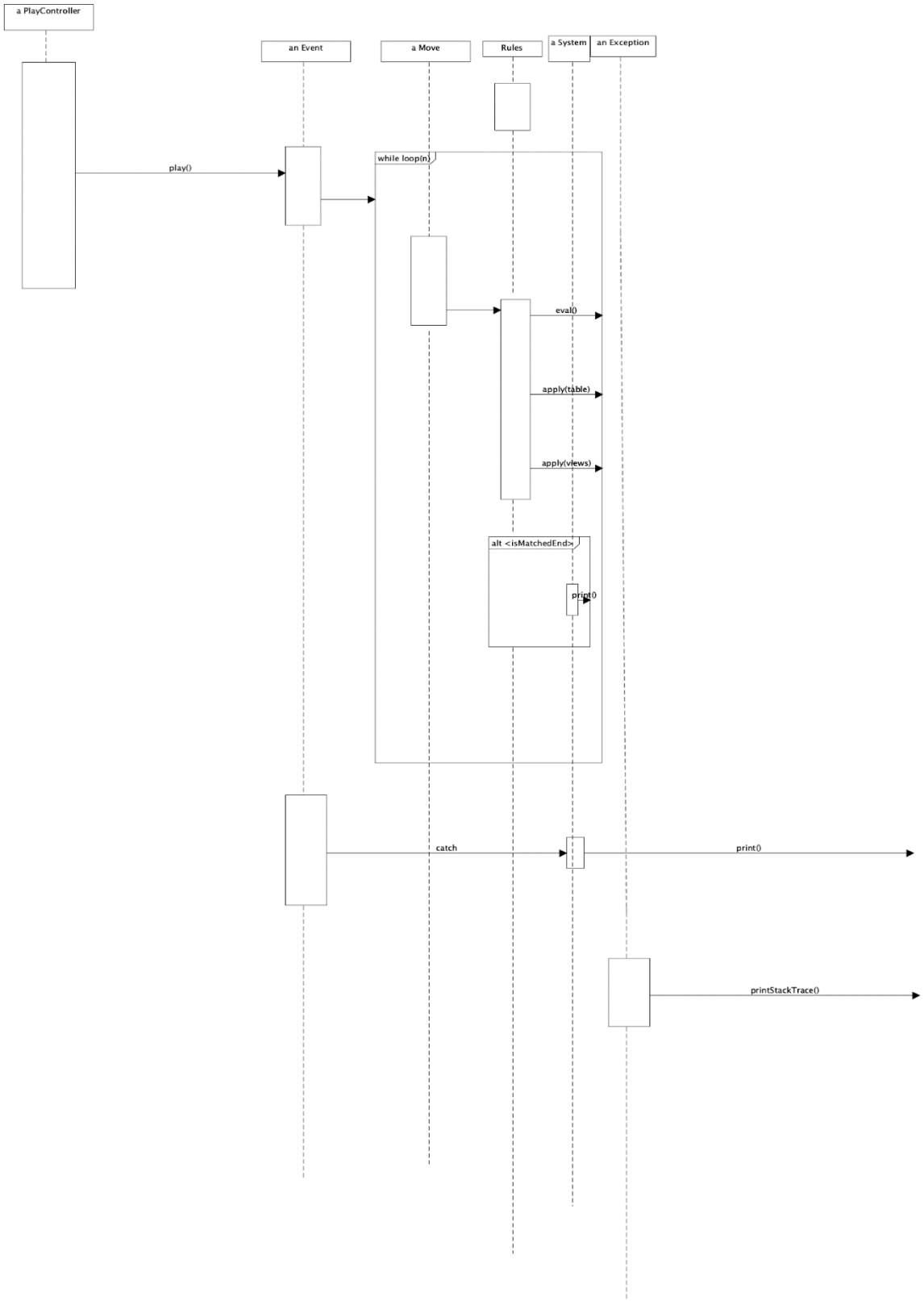
Our selected change is to "*Support the host being able to select from multiple rule variations of a game.*" To implement such changes, multiple refactors to the given code will be necessary. For example, consider the Class Diagram below for `PlayController.java`. As it is currently designed, it takes in a single `Rules` to be used. Rather intuitively, if we want the game to be able to select from *multiple* rule variations, then there are two main "paths of attack" to implement this. Either find a way s.t. these Rules can be defined by the host before each game or take in a `List<Rules>` instead of a single Rules s.t. the host can simply select from one of `n` rule options before a game starts.

PlayController Class Diagram




```
12  public class PlayController {
13
14      private InBoundQueue inQ;
15      private Rules rules;            // This line needs to be refactored
16      private List<Rules> rulesList;  // to be something like this.
```

More generally, a thorough understanding of how `Rules` is integrated within the project will be crucial. `Rules` are used throughout the system. `PlayController.java` has already be mentioned, but some other notable locations are in `MatchController.java`, `Rules.java` (intuitively), `P52Rules.java`, `RulesDispatchBase.java`, `P52MPGameFactory.java`, and `P52SPGameFactory.java`.



Our team decided to use the `PlayController` to implement the rule variation. Before calling `play()`, we need to figure out the rules first. We can first create a list of rules that are selected from the host before the event takes place. Then we will assign the selected rules to the "`rules`" instance/global variable in the `PlayController` class. Now when we call "`Move move = rules.eval(nextE, table, player);`", we will be checking with the selected rules instead of the general rules.

So, in order to select the rules via user interface, the program needs to iteratively ask for/request one rule at a time from the host and make sure those rules are suitable for the game. If not, then the program tells the host that the rule cannot be accepted. If the host finishes the selection, then the rule will enter the `play()` method. In the `Rules` class or `RuleDispatch` class, we can also add more methods such as `applyForVariation()` or something similar for applying the different rules into the program and saving them in the rules list.

Our team also feels the need to include the impact on the diagrams too, so let's use the sequence diagram for the PlayController class as an example. In order to achieve the rules variations, we will have to have an arrow pointing from the `PlayController` to `Rules` object and call a new separate method called maybe `setUpRules()`. And this is done before the `PlayController` object calls `play()`. Then in this new `setUpRules()` method, there will be a while loop for checking inputs from the user interface(the host). In the `while` loop, the program will either accept or reject the rule variations selected from the host(by calling `applyForVariation()`). If the rule is not applicable, then call break or throw an exception. If all rules are done, then we call `play()` and an Event object/box will be created. The rest of the sequence diagram will remain unchanged.

We chose this approach because the `PlayController` has the "rules" variable involved. We also wanted to follow the open-closed principle where we did not have to modify any existing code. We could simply just assign different rules to the list and add more methods to help detect if the host wants to set up the rules in multiple variations.

Of course, as we continue delving into the code, and potentially implementing changes, we will gain a clearer understanding of how exactly this should be refactored. If it turns out our current notion of how this should be implemented is naïve, we can go back to clean up the design to match our updated knowledge.