Step 1:

a. Is this MVP really minimal?

It is minimal as the game is simplified down to the point that it gives a potential client an understanding of the direction the final product is headed in while maintaining its functionality. Details such as graphics, sound effects, menus, etc are left out as they are only there to build upon user experience and not serve an important purpose.

b. Is this MVP really viable?

It is viable as there are enough features included such that it is a good enough representation of the client's wants. While several features are removed, such as the ability to select which dice are rerolled, it is good enough as it gives a client a feel for how the game will play out in the final product.

c. Can some features be simplified

F03 can be simplified slightly as being able to choose the number of games to play feels a bit unnecessary. Since this is not really a playable game of Piraten Karpen but rather a simulation of how it is played, it may be wiser to simply keep the number of games played as a constant instead of user input, just to keep it as simple as possible.

Step 2:

a. Which part of your code do you consider to be technical debt?

The random selection of dice to reroll is technical debt because it's a quick and easy way to get the job done instead of implementing actual strategies. Having the game specifically simulate 42 games is also technical debt because ideally we would like to choose how many to simulate through a command line argument or user input.

b. Do you think your features were the right size? Too big? Too small?

Some features are a bit too small, like roll dice and roll eight dice. These two are essentially the same thing but with a few more lines of code for eight dice so they can be combined into one. Other than that, each feature is the right size because they're all important to the MVP and serve one purpose.

c. Is it worth tracking the realization of each feature in the backlog? As a tag in the VCS?

The backlog is great for tracking progress of individual features as they are being implemented. It helps to give an understanding of what's done, what's in progress, and what needs to be started. The "Blocked" status is also useful because some features rely on others so it helps to know what order to implement them in. Tags are also useful for

feature tracking but since they can only be pushed to once, it's better to use them to mark when a feature is fully completed.

d. Does it make sense to sacrifice quality in the long run? For short-term?

Maintaining quality is important in the long run because our end goal is to deliver a product that the client is happy with and we want to ensure the code is well structured and easy to follow so anyone reading it or working on it in the future can understand it. In the short-term it's less important because we want to make sure our product is functional and serves its purpose before we make it more appealing.

e. What are your plans to reimburse this debt during the next iteration?

I will separate things into classes to better structure the code, adding objects for players, strategies, etc. I will also generalize code more, for example, I would make it easier to play with more than 2 players without having to add a lot of code.