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.

Step 3:

a. What are the pros and cons of delivering the MVP first and then repaying your debt?

Delivering the MVP first allows you to give the client an idea of where the project is headed without spending too many resources. It reduces the amount of time you spend on the first drafts in case the client is unhappy or wants changes. That way you dont lose too much progress if changes are needed.

The downsides are that a lot of time is spent fixing the horrible code you wrote to deliver the MVP quickly instead of spending that time implementing more features.

b. Could you have developed your MVP with less technical debt? If so, what prevented you from doing so?

Yes, I could have developed my MVP with less technical debt by structuring my code better. I could have implemented more classes and made the code more generic instead of implementing it to work in a specific way. What prevented me from doing this was time constraints. Doing all of that work to fix my code in step 3 took a lot longer than expected so it was a good idea to leave most of those fixes out for step 2.

c. What are the pros and cons of using logging mechanisms instead of print statements?

Logging statements are better for keeping track of the program's outputs because it has more information and functionality than normal print statements. For example, the log messages are usually prefixed with the current date and time which is useful for tracking progress. Log messages also have multiple output styles such as information, debug, alert, etc. They can also be saved to a log file instead of displayed on a terminal where the information is volatile.

d. Introducing Log4J is not a feature. It does not add business value to the product. How could we track the progress of such development activities?

Github releases are a good way to track development progress. At each iteration, you can create a new release to highlight the completed features and let others download and test the code.

a. Were you able to deliver all the features during this iteration? If yes, was it challenging? If not, how did you decide which features could be pushed to the next iteration?

I was unable to deliver the feature where strategies can be selected through the command line, as there were problems with strategy instances not being updated. I left this feature for last because it required the combo strategy to actually work before I could select it, and the combo strategy required the new scoring mechanism which was implemented first. This is an easy feature to push to the next iteration as I can manually change the strategy through code.

b. What is your status in terms of technical debts?

I've cleared most of the technical debt from previous iterations by adding more classes, implementing methods to be more general, etc. Whenever I saw one class getting too large or convoluted, I moved code to a new class to make sure it stays readable. There is still some remaining debt as there are multiple features coming in later iterations which haven't been considered yet.

c. How did the object orientation of your code support the introduction of the new strategy and scoring mechanism?

There are several objects in the game which helped with updating code as I could easily add methods to an existing class which is related to the function instead of digging through long, unorganized classes or large main methods. For example, I had a function in the Player class to handle score counting. A few quick updates to it allowed me to score players based on combos. I also created a strategy interface which I could use to implement multiple strategies and also users to pass any class implementing strategy to the Player constructor. The strategy interface contains functions for deciding whether or not the player can or should reroll, and selecting which dice to roll if they can.