Product planning

In this document you will find our product backlog and our definition of done

Product backlog (MoSCoW method)
We created our product backlog using the MoSCoW method:

Functional Requirements
Must haves

- The game must be run on a pc.
- For the pc application:
    - Upon the start of the game the user will be shown a main menu.
    - The main menu will have a "Exit" button button that exits out of the game
    - The main menu will contain a button: "Host Game".
    - When the user presses the Host Game button, he will get put in a lobby screen.
    - The lobby screen will contain a list of all devices connected to the lobby.
    - Once exactly four Android devices are in the lobby and an Oculus Rift is connected, a "Start Game" button will show up on screen.
    - When the user presses the "Start Game" button, the game will start for all connected devices.
    - The lobby screen will contain a button "Back to main menu".
- For the android app:
    - Upon opening the app the user is shown the main menu
    - The main menu must have a "Exit" button button that exits out of the app
    - The main menu must have a "Play" button that links to a "Choose Game" screen
    - The "Choose Game" screen contains a list of selectable hosted games within the LAN network
    - The "Choose Game" screen contains a button "Join Game".
    - The "Join Game" button in the "Choose Game" screen lets an android device player join a game if one is selected.
    - The "Join Game" button should become clickable once a lobby has been selected.
    - When the user presses the "Join Game" button, it will be put inside the lobby screen.
    - The lobby screen will contain a "Back to main menu" button that when clicked will return the user to the main menu.
    - The lobby screen will contain a list of all devices connected to the lobby.
    - The game starts when the host of the game presses a "Start Game" button.

Note: The user using the Oculus Rift and the users using an Android device will from now on be referred to as the "commander" and "carrier" respectively.

- Upon start of the game, the commander is put on top of a small platform.
- The commander can perceive the world through the Oculus Rift in first person.
- Each of the carriers is standing under one of the corners of the platform, making it look like they're actually carrying the platform for the commander. The carriers DO NOT perceive the world.
- The platform will always move forwards at a constant speed, which will depend on the difficulty of the game.
- The platform can slightly bend left and right, depending on how the carriers tilt their Android devices.
- Depending on how the carriers tilt their Android device, the platform will get tilted in the vertical axis as well. Example: There are two carriers tilting their device; one in the top right corner, the other in the bottom right corner. The one in the bottom right tilts his device left and the carrier in the top right tilts his device right. The platform will then get a slight slope where the bottom right corner is positioned lower than the top right corner. This gives the platform a certain slope.

- The world in which the users are located is shown (to the commander) as an infinitely long road containing walls on the edges.
- The road will contain several obstacles that the carriers should try and avoid.
    - Examples of obstacles could be a road block on one side of the road, a large random pit, a traffic sign, etc.
- Effects of hitting an obstacle can vary depending on the obstacle. A tilt in the platform for example, or damage done to one or more of the carriers.

- The game should have checkpoints after a set distance, you should be able to start the next game from this checkpoint if you have reached it before, this is to skip the "easy" early game.

- Each carrier has a certain amount of health.
- Health can get reduced from certain effects happening in the world (such as hitting an obstacle).
- When a carrier has no health left, the player will be "knocked-out".
- When a carrier is "knocked out", the player will not be able to do anything in the game, until the player recovers.
- When a carrier is "knocked out", the player will be able to recover by rapidly tapping the screen of his/her smartphone.
- When a player recovers, he/she will be able to do everything the same way as before the knock-out.
- When two carriers are knocked-out at the same time, the platform falls over and the game is lost.

- If the platform gets too steep due to the carelessness of the carriers, obstacles, the platform falls over.
- If the platform falls over, the commander falls over with it (giving a real traumatizing experience for the Oculus Rift user), and the game is over (everyone is put back in the lobby).

- The carrier will be able to see their health on his/hers Android device.
- The carrier will be able to see the platform on their devices, so that they can use it to balance the platform more. The position of the carrier in relation to the platform should also be visible. The platform is the only thing the carriers can perceive from the world.
- The carrier will be able to dodge obstacles by jumping or ducking. This movement will be detected via movement detection in the "normal" world.

<u>Should haves</u>

- When one carrier gets "knocked out", the platform should become much harder to handle for the remaining carriers.

- The game should have enemies:
    - Enemies appear on the road.
    - Enemies deal more damage the further the platform gets along the path
    - The commander is the only one who can visually detect enemies and see where the enemies go
    - Enemies have 3 places relatively from each carrier where they can spawn and attack that carrier from, each on a 90 degree angle from each other. (The fourth place is under the platform and would not be able to be detected by the commander)
    - When an enemy get killed, it disappears from the game.
- The carriers should be able to defend themselves from enemies:
    - Carriers each have a half circle on their screen which indicate the spot they can attack
    - The half circle is inverted for each set of carriers on each side (because the missing part of the circle indicates where the enemy can't be located, which is under the platform)
    - A carrier can only attack once every 3 seconds, this is to prevent carriers from spamming each direction to kill incoming enemies. They should get input from the commander on where they should attack.
- The game should have an options menu which can be accessed via the pc on which the game is run.
- The game should have a randomly generated world, which consists of pre made parts of the world put together. A part could for example exists out of 2 walls, a floor, enemies and an obstacle.
- The game should get harder as the platform gets further along the path. Harder means more obstacles, more enemies and higher speed.

Could haves

- The game could have multiple game modes.
- The “Choose Game” screen in the android app could have search refining options for hosted games.
- The game could have various weapons to choose from for the players
- The game could have a possible action of making a 90 degree turn by letting the mobile users switch rules (left turn: player right in front -> right in the back)
- The game could have different color schemes to change the atmosphere.
- The road of the game could have angles and corners to make for a more interesting and immersive world.
- The death animation of the enemies could be added.
- The android device users should have an option to mute music and sounds from their devices, this should be done by 2 toggle buttons in the top right corner of the device.

Won’t haves

- The game won’t have support for a different amount of mobile users.
- iOS and Windows phone support

Non-Functional Requirements

- Usage of one Oculus Rift
- Usage of four android phones; minimum android version 4.0
- Usage of the jMonkey sdk
- Communication between devices happens over a LAN

Definition of Done (backlog items, sprints, releases)

In our definition of done, we will define under which conditions we consider some part of our project “done”. We will do this for every feature, sprint and for the final release.

We consider a feature to be done when:

- The code and tests for the feature all compile and do not contain any errors.
- It is fully implemented and fully tested.
  - The testing is done through both unit testing and end-to-end testing.
  - Obviously all tests for that feature have to pass.
  - The test coverage of the feature is at least 70%, or a sufficient explanation of why that feature can’t be tested using unit / end-to-end tests.
- Both the code and the tests are fully documented with javadoc and comments.

We consider a sprint to be done when:

- The features for the sprint are all completed and tested as described above.
- User tests are passed. That way we know for sure that there are no bugs in the code when the features are all merged (like with a pull request or an append to already existing code).

We consider our final release to be done when:

- The code doesn't contain any errors any longer.
- All code is completely documented with javadoc and comments.
- All must-haves in this document are implemented.
- Some should-haves in this document are implemented.
- The game is fully tested with users to make sure there are no bugs in the actual gameplay.
- All of our tests pass.
- We reach a total test coverage of at least 70%.

References:

1. Kenneth, S. Rubin, "Essential Scrum: A Practical Guide to the Most Popular Agile Process"