Methodology

1 Roles

1.1 Roles and Responsibilities

1.1.1 Team Lead

The Team Lead is responsible for overseeing the project, ensuring that all teams are working correctly for the project objectives, facilitates communication between teams, and address any issues that may arise.

1.1.2 Architect

The Architect is responsible for the design of the software solution for the project. He ensures that the architecture meets both the needs of the project. The Architect collaborates closely with the Development Teams to make sure that the implementation aligns with the architectural vision and standards.

1.1.3 Scrum Master

The Scrum Master facilitates the Agile process within the team. He organizes and conducts Scrum ceremonies such as Sprint Planning, Daily Stand-ups, Sprint Reviews, and Sprint Retrospectives.

1.1.4 Product Owner

The Product Owner is responsible for defining the product vision and managing the product backlog. He prioritizes tasks, ensuring that the team delivers features that provide the most value. The Product Owner communicates the project requirements and changes to the development teams.

1.1.5 Development Team

Games Team The Games Team focuses on developing example games such as a tic-tac-toe and a Pac-Man. They work on implementing game logic. The Games Team collaborates closely with the Console Team to integrate input and output support.

Console Team The Console Team is responsible for developing the support for input events like keyboard and mouse, as well as output support for audio and graphical user interfaces. They ensure that the platform can run the games.

1.1.6 QA Team

The QA Team ensures the quality of the software solution by conducting rigorous testing. They identify and report bugs. The QA Team works closely with the Development Teams to ensure that issues are resolved.

1.1.7 DevOps Infrastructure Team

The DevOps Infrastructure Team is responsible for setting up and maintaining the development and production environments. They manage the continuous integration and continuous deployment (CI/CD) pipelines.

1.2 Team Lead & Architect

Team Lead: Fabian Romero Claros

Architect: Gabriel Santiago Concha Saavedra

1.3 Sprint 0

Product Owner: Emanuel Galindo Corpa

Scrum Master: Jose Luis Terán Rocha

1.3.1 QA Team

- Ronaldo Miguel Ángel Mendoza Mallcu
- Sebastián Barra Zurita
- Jhael Arce Chavez

1.3.2 Dev Team

- Fabian Romero Claros
- Gabriel Santiago Concha Saavedra
- Josue Mauricio Prado Camacho
- Luis Enrique Espinoza Vera
- Luiggy Mamani Condori
- Axel Javier Ayala Siles
- Victor Leon Villca Silva
- Alex Paca Meneses
- Jose Luis Terán Rocha

1.4 Sprint 1

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

1.4.1 QA Team

- Jhael Arce Chavez - **QA Lead**
- Sebastian Barra Zurita
- Ronaldo Miguel Angel Mendoza Mallcu

1.4.2 Dev Team

Front End Team

- Victor Leon Villca Silva - **Lead**
- Emanuel Javier Galindo Corpa
- Luis Enrique Espinoza Vera
- Josue Mauricio Prado Camacho

Back End Team

- Alex Paca Meneses **- Lead**
- Gabriel Santiago Concha Saavedra
- Axel Javier Ayala Siles
- Fabian Romero Claros
- Luiggy Mamani Condori
- Jose Luis Terán Rocha

1.4.3 DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

1.5 Sprint 2

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

1.5.1 QA Team

- Ronaldo Miguel Angel Mendoza Mallcu - **QA Lead**
- Jhael Arce Chavez
- Sebastian Barra Zurita

1.5.2 Dev Team

Front End Team

- Emanuel Javier Galindo Corpa - **Lead**
- Victor Leon Villca Silva
- Luis Enrique Espinoza Vera
- Josue Mauricio Prado Camacho

Back End Team

- Axel Javier Ayala Siles - **Lead**
- Gabriel Santiago Concha Saavedra
- Alex Paca Meneses
- Fabian Romero Claros

- Luiggy Mamani Condori
- Jose Luis Terán Rocha

1.5.3 DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

1.6 Sprint 3

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

1.6.1 QA Team

- Josue Mauricio Prado Camacho **- QA Lead**
- Luis Enrique Espinoza Vera

1.6.2 Dev Team

Games Team

- Fabian Romero Claros **- Lead**
- Ronaldo Miguel Angel Mendoza Mallcu
- Jhael Arce Chavez
- Victor Leon Villca Silva
- Emanuel Galindo Corpa
- Sebastian Barra Zurita

Console Team

- Luiggy Mamani Condori **- Lead**
- Gabriel Santiago Concha Saavedra
- Alex Paca Meneses
- Axel Javier Ayala Siles
- Jose Luis Terán Rocha

1.6.3 DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

1.7 Sprint 4

Product Owner: Alex Paca Meneses

Scrum Master: Ronaldo Miguel Angel Mendoza Mallcu

1.7.1 Dev Team

Games Team

- Fabian Romero Claros - **Lead**
- Ronaldo Miguel Angel Mendoza Mallcu
- Jhael Arce Chavez
- Victor Leon Villca Silva
- Emanuel Galindo Corpa
- Sebastian Barra Zurita

Console Team

- Luiggy Mamani Condori - **Lead**
- Gabriel Santiago Concha Saavedra
- Alex Paca Meneses
- Jose Luis Terán Rocha
- Josue Mauricio Prado Camacho
- Luis Enrique Espinoza Vera
- Axel Javier Ayala Siles

1.7.2 DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

2 CEREMONIES

2.1 Scrum of Scrums

We will implement a Scrum of Scrums approach to facilitate coordination among multiple teams. This will involve regular meetings among the team leads to discuss progress, dependencies, and impediments that affect the larger project.
We will work using the agile framework, so our ceremonies will be the following:

2.2 Daily Lead Stand Up

To ensure coordination among team leads, we will have daily stand-up meetings specifically for the leads of each sub-team, the team lead, the PO and the architect.. These meetings will focus on high-level updates, strategic decisions, and any cross-team dependencies or issues. The meeting will be on Tuesdays and Thursdays after finishing the daily stand ups of all the teams.

2.3 Sprint Planning:

Every Monday will be our meetings where Dev and QA Team will follow the next points:

- Use of **Poker Planning** to estimate the points of each US.
  - The story points will be defined by the complexity of development and QA team.
- While we are defining our **Sprint Goals** and prioritize of new US to work
  - During the meeting, we will clarify any ambiguities or doubts about the US, which can conduce to a refinement of the US that is considered as **Backlog Refinement** or **Backlog Grooming.**

2.4 Sprint Duration:

- As a team, we define our sprint of 1 weeks.

2.5 Daily Stand Up:

Daily stand-up meetings will be held by each sub-team to communicate progress and discuss any roadblocks. The schedule for these daily stand-ups will be set and agreed upon by each sub-team. The Scrum Master must attend all daily stand-up meetings.

2.6 Demo - Delivery:

Every Friday will be our demo about the US worked during the Sprint.

- Having the demo of the product and the report and metrics of QA Team during the Sprint.
  - In the final demo will present the entire product developed with some metrics about US and also, the whole metrics by QA Team.

2.7 Sprint Review:

Every Friday will take into account the following points:

- Review of the work that has been completed during the sprint.
  - What items have been completed?
  - Do we some US for Carry Over?

- **Review of the Product Backlog:**
  - Current state of the product backlog, what items remains.
  - Changes are made based on the feedback received by Dev and QA Team and any changes according PO needs.
- The team and PO will collaborate on what to work on next, helping to prioritize the next sprint's backlog.
- We will review what US did not go well in the Sprint and how we will fix it.

2.8 Sprint Retrospective:

Every Friday when finishing the Sprint, where we are gonna talk about what went right, what went wrong and what can be improved for our next iteration. Having feedback from the sprint review.

- While the meeting is happening, we will work on our **Action Items**.
  - Identifing things we should start doing, stop doing, and continue doing in future Sprints, as well as specific action items to implement these changes.

3 ARTIFACTS

3.1 Product Backlog

We maintain a prioritized list of all the features, enhancements, bug fixes, and other work needed to complete the project. It's our single source of truth for what needs to be done.

Link: Product Backlog in Taiga.

3.2 Sprint Backlog

We select a subset of the Product Backlog items for a specific Sprint, along with a plan for how to deliver them. It's created during the Sprint Planning meeting by the Development Team and serves as a guide for our work during the Sprint.

3.3 Burn-Down Chart

We use a representation of the amount of work remaining in the Sprint. It shows our progress towards completing the Sprint's goal and helps us track whether we're on track to finish all planned work by the end of the Sprint.

3.4 Impediment Log

We keep a record of the impediments or problems of each team member per day It's used during our Daily Standup meeting to identify any issues that need to be addressed.

Link: Impediment Logs Sheets.

3.5 Start-Stop-Continue-Action Items

We identify things we should start doing, stop doing, and continue doing in future Sprints, as well as specific action items to implement these changes. It helps us reflect on our process and make improvements for future iterations.

Link: Start-Stop-Continue-Action Items in Miro.

4 DOR/DOD

4.1 Definition of Ready

- The story should have a clear description that indicates what it hopes to achieve and why it is important.
- Clear criteria should be established indicating when the user story will be considered complete.
- The necessary resources (such as people, tools) must be available to complete the story effectively.
- Before a story can be considered ready for development, it must have been prioritized and estimated.

4.2 Definition of Done

- Code reviews must be performed by at least one and max two team members.
- All issues identified during the code review should be fixed.
- Unit Tests must be completed and successfully passed before a task is considered ready for delivery.
- All methods and classes from the library must be documented.
- The documentation in the library provided should be clear, concise, and adequately describe the functionality of the code.
- The architecture defined by the team must be followed, including established design patterns and conventions.

5 WORKFLOW AND GITFLOW:

5.1 Workflow:

5.1.1 Availability / Support Time:

- Monday - Wednesday - Friday: 14.30 - 16.30
- Tuesday - Thursday: 14.30 - 17.30

5.1.2 Developer:

- **Identification of User Stories (US):** the team will be in charge of identifying the User Stories that must be developed. These are short, simple descriptions of a feature told from the end user's perspective.
- **Prioritization of User Stories:** based on several factors, such as the value they provide to the end user, the difficulty of implementation, the dependencies between them, etc.
- **Assignment of User Stories:** assignment of each US to a member of the development team. This should be done taking into account the skills and abilities of each team member, as well as their current workload.
- **Taiga Dashboard Update:** at each stage of development, we constantly update our Taiga dashboard to reflect progress.

5.1.3 Code reviews:

- **Peer review:** when a developer finishes a task and uploads it to the repository (makes a commit), another developer on the team should review the code. Aiming to detect errors and improve code quality.
- On an ongoing basis, the development team leader will give feedback to the development team regarding the code that is considered most critical, or make a final review before the code is merged into the master branch.
- **Code review automation:** code review tools will be used to automate part of this process such as **CI/CD**. Helping detect code style issues, programming errors, and other code quality issues.

5.1.4 Ready for QA:

- Once a US is finished, it goes to "Ready for QA" status, we notify the QA team and wait for their feedback.
- In case issues are detected, we proceed to correct them and send the US back to the **TEST** branch. If there are no problems, the US goes to production.

5.1.5 Quality Control:

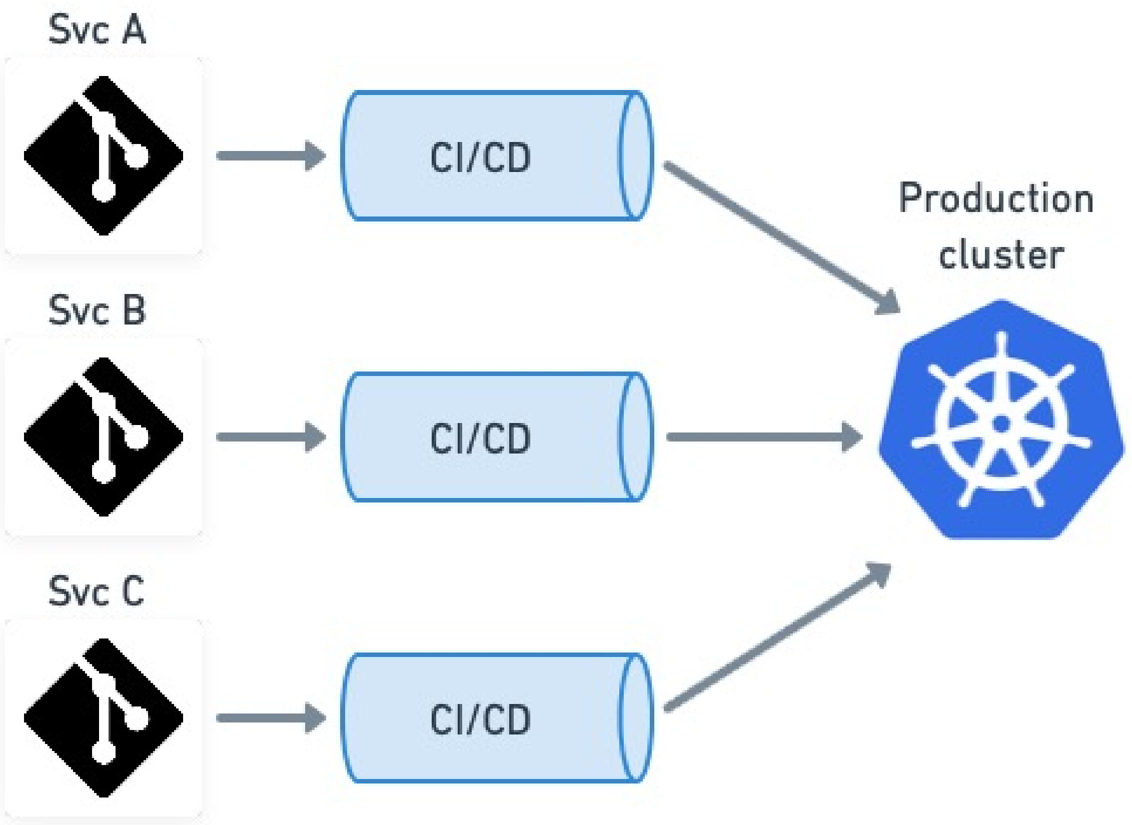
- We assign a US that can be tested to a member of the QA team.
- In the QA phase, they will report the bugs found in the different US and will be notified in Taiga for further information on the section **Issues.**

5.1.6 Production:

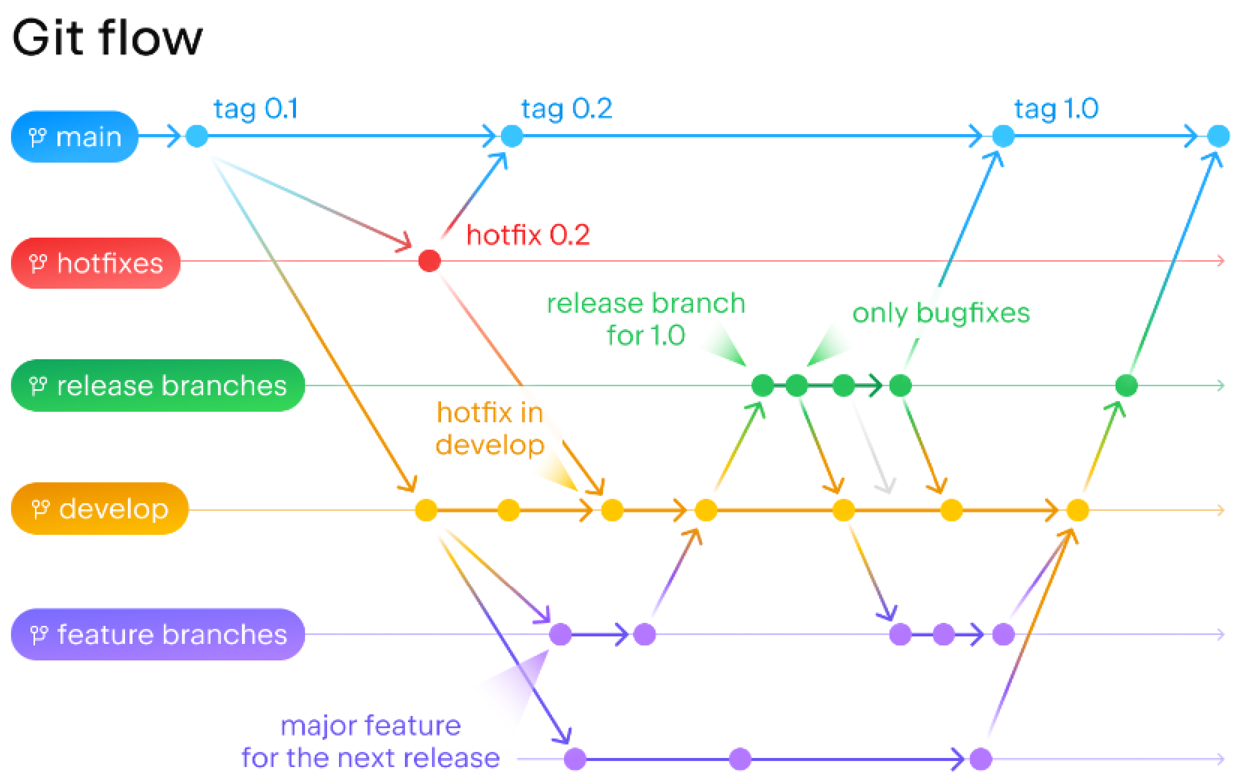
- After QA approval, we merge the US to production and continue to the next US.

5.2 Gitflow

We want to separate the different modules into different repositories to better control integration and continuous distribution (CI/CD) independently.
The repositories will be independent from each other but will have a logical relationship, so a Community will be created to add that relationship.
There is a Git Workflow called Multi-Repo where this same approach is used, although it will not take all the concepts such as the use of macros, but we will retain the organization by logic and independent development.



For the moment, each repository will manage the GitFlow structure, and a small modification will be made by adding a testing branch for the QA team.



- The `main` branch is for production code only.
- The `develop` branch is for development code.
- `feature` branches are created from the `develop` branch.
- `hotfix` branches are created from the `main` branch.
- `release` branches are created from the `develop` branch.

5.2.1 Conventional Commits:

A specification for adding human and machine readable meaning to commit messages

1. **fix:** fixes a bug or defect.
2. **feat:** adds a new feature or functionality.
3. **build**: changes related to the build system or dependencies.
4. **chore:** maintenance or housekeeping tasks.
5. **ci:** changes to continuous integration.
6. **docs:** documentation changes.
7. **style:** code appearance or formatting changes.

8. **refactor:** significant code restructuring without behavior change.
9. **perf:** performance improvements.
10. **test:** changes related to testing.

5.2.2 Pull Request structure:

- **Pull Request Title:** Title should be clear and concise, and should accurately reflect the purpose of the change.
- **Pull Request Description:** The description should provide details about what changes were made and why. You should include any relevant context that helps understand the change.
  - We will use the **Pull Request Template** and answer the following 3 questions:
    * What did I do?
    * How did I do it?
    * Why did I do it?
- **References to Issues:** If the change is related to an issue in your issue tracking system (such as Jira, GitHub Issues, etc.), it should be mentioned in the description. This will help connect your pull request to the problem it is solving.
- **Request Reviewers:** Ask one or more members of your team to review the pull request. They will provide feedback and suggest improvements if necessary.