Game Engine Documentation

Coffee Time

Contents

1 Profile

1.1 Introduction

Game development requires programmers to either learn how to use a complex game engine or low level programming languages. Our solution aims to provide new game developers with a platform where they spend more time developing a game than managing graphics or I/O.

1.2 Problem Description

Game development requires the usage of low level programming languages or complicated game engines. There is a need for a high level programming language that can be used to develop games without having to manage input device peripherals or a graphical user interface.

1.3 Objectives

1.3.1 General Objective

Create a software solution, a platform that allows high level language programmers to develop games without worrying about a graphic user interface or I/O and instead spend more time developing the game.

1.3.2 Specific Objectives

- Example games:
  - A simple tic-tac-toe PRD Tictactoe
  - A generic Pac-Man PRD Pacman
- Input peripherals support
  - Keyboard
  - Mouse
- Output support
  - Audio
  - Graphical User Interface
- Game Development Library
  - This library will provide necessary tools for game development.

1.4 Scope and Limits

1.4.1 Scope

- Java developers that want to develop games.

1.4.2 Limits

- Hardware
  - A GPU that supports Vulkan, since version 1.3

- Miscellany
  - Games only can be played locally.
  - Only 2D graphics.
  - To run the game, the end-user has to have basic terminal knowledge.
  - The library will be written in Java "JDK" 17.
  - Resolution for Games will 800x600

2 Theorical Framework

2.1 Technology justification

Has a team we divided by front-end and back-end, front-end refered to games development and the library, for backend we manage screen and console, console has a interpreter and screen has the peace that contains GUI, input/output and sounds.

2.1.1 Frontend

The team agreed to use Java as a frontend language, taking into account that it is a high-level language, from an object-oriented paradigm that the team already knows and has worked with before.

Language	Must be an OOP language	Must provide a testing suite
Java	YES	JUnit 5
C	NO	UNITY

2.1.2 Backend

Rust

- Is a Low-Level and compiled language
- "Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages." – Rust-Performance
- Graphs management
  - Window handling:
    - Wayland
    - X11
  - Graphics
- I/O access
  - Rust provides a variety of libraries for operations like listening for keyboard or mouse events.
- Foreign Function Interface Support
  - **Java:**

    * JNI Bindings for Rust is a library that allows us to:
        · Implement native Java methods for JVM in Rust
        · Call Java code from Rust
        · Embed JVM in Rust applications and use any Java libraries
  – **C:**
    * For static linking, a build process must be performed for the C code based on a CMakeLists.txt configuration, where it uses the cmake dependency to create a library.
- Technology justification Doc

2.2 Game Communication

2.2.1 Description

To observe o understand the flow of information, lets begin with the game. The game is an implementation in Java that must define the attitudes, events that the game will have, the programming of when you win and when you lose, what happens if a character collides with something in its environment, etc..., it is the responsibility of the game. take care of everything related to the logic of the game and its actions such as creating its resources such as sounds or images, for which the game itself must implement those events, Obeying the contract.

The contract is a library implemented by the team, we are based on and take JavaFX as an example, a graphic library that in order to use it requires you to create certain components for its use, now we call this library within the team bridge since it works as a bridge between the game and console.

This communication works through JNI, which as we pointed out before would be the method used to communicate both languages based on the successful results of the spike referring to this implementation.

Once the information reaches the console, it will be processed by the intermediaries, the input/output intermediate, which will send the information through sockets to the screen.

Screen also has the sockets to guarantee communication with the console, it is also responsible for managing the RUST graphic library, which in this case we use ”nannou” as a graphic library, in terms of GUI (sprites), Input (keyboard and mouse), output (sounds).

For how it works lets think about a game in this case Pacman, we collide with a ghost, this event is handled by the game the ”what will happen when it collides” the state of the game is updated, since Pacman is removed from the screen by who has lost a life, so Pacman’s image must be updated on the screen, that information flows through the previously described flow to reach the screen as an update of Pacman’s image based on his coordinates

- More info on: Game Communication
- More info on: Architecture
- More info on: Actual Architecture PR

3 Methodology

3.1 ROLES

3.1.1 Roles and Responsibilities

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.

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.

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.

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.

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.

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.

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.

3.1.2 Team Lead & Architect

Team Lead: Fabian Romero Claros

Architect: Gabriel Santiago Concha Saavedra

3.1.3 Sprint 0

Product Owner: Emanuel Galindo Corpa

Scrum Master: Jose Luis Terán Rocha

QA Team

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

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

3.1.4 Sprint 1

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

QA Team

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

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.1.5 Sprint 2

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

QA Team

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

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.1.6 Sprint 3

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

QA Team

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

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.1.7 Sprint 4

Product Owner: Alex Paca Meneses

Scrum Master: Ronaldo Miguel Angel Mendoza Mallcu

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.1.8 Sprint 5

Product Owner: Alex Paca Meneses

Scrum Master: Ronaldo Miguel Angel Mendoza Mallcu

Dev Team

Games Team

- Luis Enrique Espinoza Vera - **Lead**
- Fabian Romero Claros
- Ronaldo Miguel Angel Mendoza Mallcu
- Jhael Arce Chavez
- Victor Leon Villca Silva
- Emanuel Galindo Corpa
- Sebastian Barra Zurita
- Josue Mauricio Prado Camacho

Console Team

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.1.9 Sprint 6

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

Dev Team

Games Team

- Ronaldo Miguel Angel Mendoza Mallcu - **Lead**
- Luis Enrique Espinoza Vera
- Fabian Romero Claros
- Jhael Arce Chavez
- Victor Leon Villca Silva
- Emanuel Galindo Corpa
- Sebastian Barra Zurita
- Josue Mauricio Prado Camacho

Console Team

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.1.10 Sprint 7

Product Owner: Alex Paca Meneses

Scrum Master: Jose Luis Terán Rocha

Dev Team

Games Team

- Sebastian Barra Zurita - **Lead**
- Luis Enrique Espinoza Vera

- Fabian Romero Claros
- Jhael Arce Chavez
- Victor Leon Villca Silva
- Emanuel Galindo Corpa
- Ronaldo Miguel Angel Mendoza Mallcu
- Josue Mauricio Prado Camacho

Console Team

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

DevOps / Infrastructure Team

- Gabriel Santiago Concha Saavedra
- Jose Luis Terán Rocha

3.2 CEREMONIES

3.2.1 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:

3.2.2 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.

3.2.3 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.**

3.2.4 Sprint Duration:

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

3.2.5 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.

- 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.

3.2.6 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.

3.2.7 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.

3.3 ARTIFACTS

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.4 DOR/DOD

3.4.1 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.

3.4.2 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.

3.5 WORKFLOW AND GITFLOW

3.5.1 Workflow:

Availability / Support Time:

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

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.

Code reviews:

- **Peer review:** when a developer finishes a task and uploads it to the repository (makes a commit), another developer or two developers on the team should review the code. Aiming to detect errors and improve code quality.
- **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.

Ready for QA:

- Once a testeable US is finished, it goes to "Ready for QA" status, we notify the
- In case issues are detected, we proceed to correct and it goes to "Ready for QA" again. If there are no problems, the US goes to production.

Quality Control:

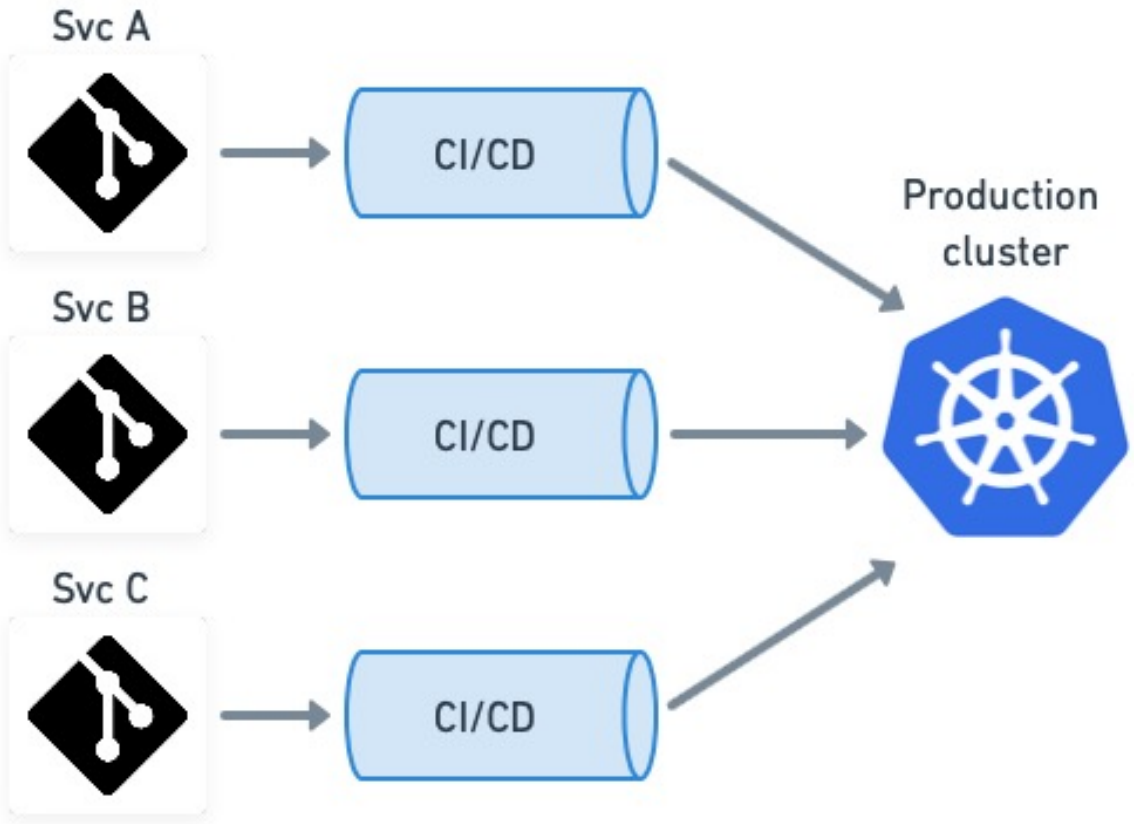
- We assign a US that can be tested to a member of the QA team.
- In the QA phase, they will report the bugs found in the different US and will be notified in Taiga for further information on the section **Issues.**

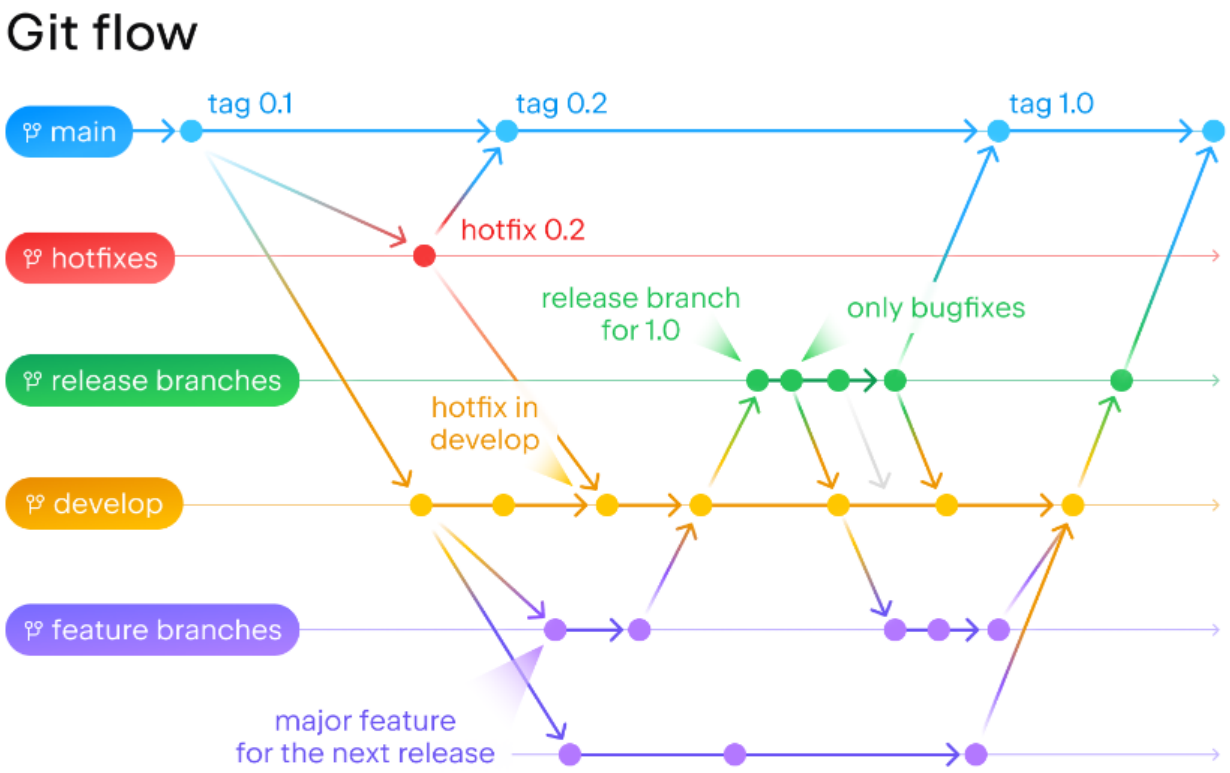
Production:

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

3.5.2 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.





- 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.

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.

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.

4 Practical Framework

4.1 Sprint 0

4.1.1 Introduction

In this sprint we were still researching concepts and tools, focused on low-level tools and languages for the console, also how to communicate console and game.

4.1.2 Spikes

We focus on the need to investigate and reduce uncertainty before committing to the implementation of concrete solutions.

LLVM For LLVM, the research was concerned with finding a way to communicate a high-level language such as Java with a low-level language, at this point it was open to C, C++ or others, LLVM was an option that, although it fulfilled its ability to compile a high-level language to a low-level one, after doing the Spike we realized that it was more related to compilers, a set of tools to be able to make compilers even of industrial size, although we found something similar to what we were looking for with a tool developed with LLVM. "DragonEgg" But it is deprecated, the documentation is scarce and for the time of the project we did not see it optimal, it was discarded.

JNI For JNI, which is Java Native Interface, a Spike was carried out to see its viability applied to a context close to the project, seeing what it could support and what not, to see its limitations, and this was the option that was taken for the project due to that the spike demonstrated effective communication between Java and Rust, so this concept was continued.

4.1.3 POC's

There POC on this Sprint its anexed on Product Backlog. The results of the POC its the development of a technichal justification:

- Spike JVM: Justification

4.1.4 Technical Justification

During Sprint 0, the team focused exclusively on conducting Spikes instead of developing POCs (Proof of Concepts). This decision was made based on the following reasons:

Reduction of Technical Uncertainty The project presented multiple areas of technical uncertainty that needed to be explored and understood before moving on to practical implementation. The Spikes allowed us to investigate different technologies, approaches, and possible solutions, providing a deeper understanding of the challenges and opportunities present.

Feasibility Assessment Before committing significant resources to the development of POCs, it was crucial to assess the technical feasibility of the proposed ideas and approaches. The Spikes provided the opportunity to conduct brief and focused investigations, which allowed us to quickly identify viable solutions and discard those that were not, without the need to build complete implementations.

4.1.5 High-Level Architecture (HLArchitecture)

For the time of this Sprint there were no a first oficial version of the architecture, but the architecture that the SPikes and POC end of was the first version:

- Results: Architecture first version you can see it on:
- Link to C4 dsl: Visualizer

4.1.6 Product Backlog

Done Completed spikes.

- Spike JVM: Task id-2
- Spike C VM: Task id-4
- Spike C++ VM: Task id-3
- Spike Bytecode and C: Task id-5
- Spike Reflection: Task id-6
- Spike LLVM: Task id-8
- Spike Sprites Sounds Effects: Task id-9
- Spike Implicit calls in java: Task id-10
- Spike KISS, DRYand YAGNI: Task id-12
- Spike Assembler: Task id-13
- Spike web Assamebly: Task id-14
- Spike Interop communication between two programs: Task id-15
- POC: Tech stack: Task id-16

Conclussions This Sprite most work was on investigate ways to communicate a console in a low level language with the game, So all the Spikes were refered to understand a clair way to achieve that communication, all the spikes were ended on time, has a result of this spikes we choose JNI, has the way to communicate our game in Java with a console writed on Rust, also we discard LLVM, C VM, C++ VM and Assambly.

4.1.7 Epics

For this Sprint 0 we have the following epics.

- Library
- Console
- Input/output
- Graphics

4.2 Sprint 3

4.2.1 Introduction

During this sprint the team focused on the implementation of the input handler and input handler for bridge, as well as input management and sound management. The UMLs of each were assigned and based on these, implementation tasks were assigned based on the UMLs.

4.2.2 POCs

No proofs of concept (POCs) were conducted in this sprint.

4.2.3 Technical Justification

In Sprint 3 the decision was made to design the UMLs and implement them. The decision was made based on the following:

- Being able to handle keyboard and mouse events
- Hear the sounds of the game through the console.
- Communicate game with console.
- Handle the updates of the information beetwen game and console.

4.2.4 Scrum of Scrums

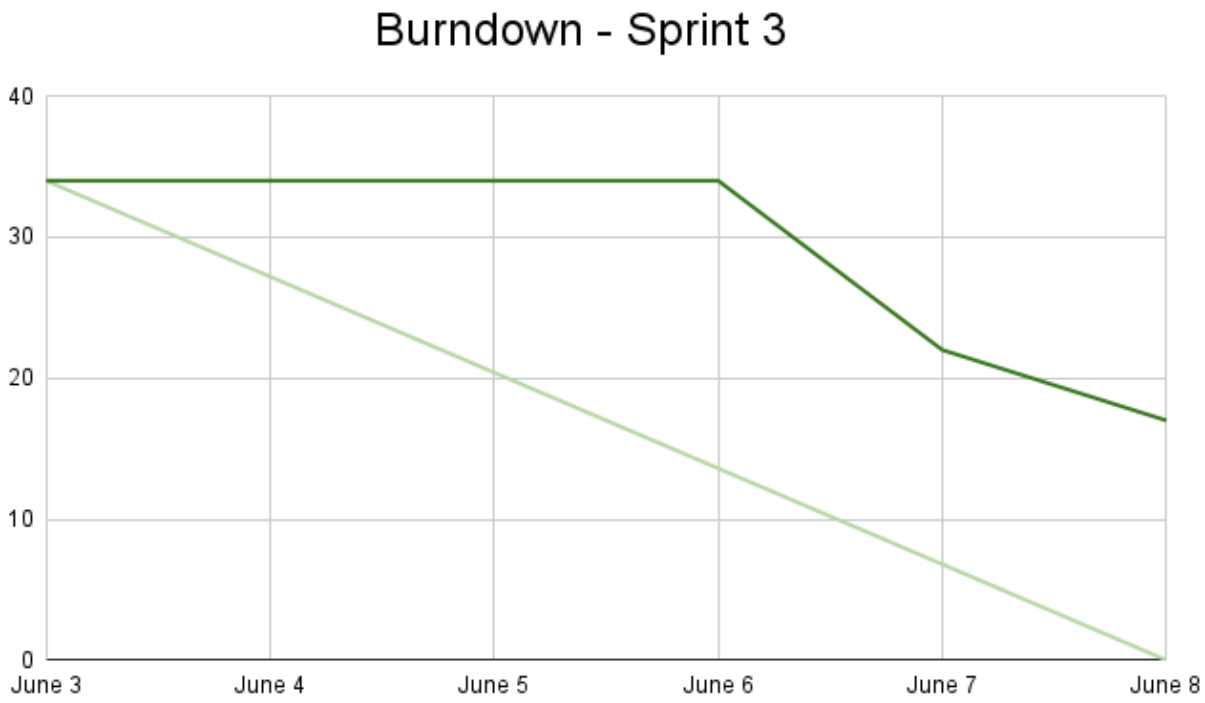
At this point, the team had reorginazed the Epics due to the new architecture changes. Having Screen, Bridge and Games as main Epics. During this sprint, while working on User Story 87, there was a refactoring in the architecture due to GUI should be running along with the game. Then, in the Lead's Daily a discussion occurred about that change, having it as an agreement furthermore, the contract (frontend-library) been terminated in its initial version.

Link: GUI Refactor - Pull Request on GitHub.

Link: User Stories of Sprint 3 on Taiga.

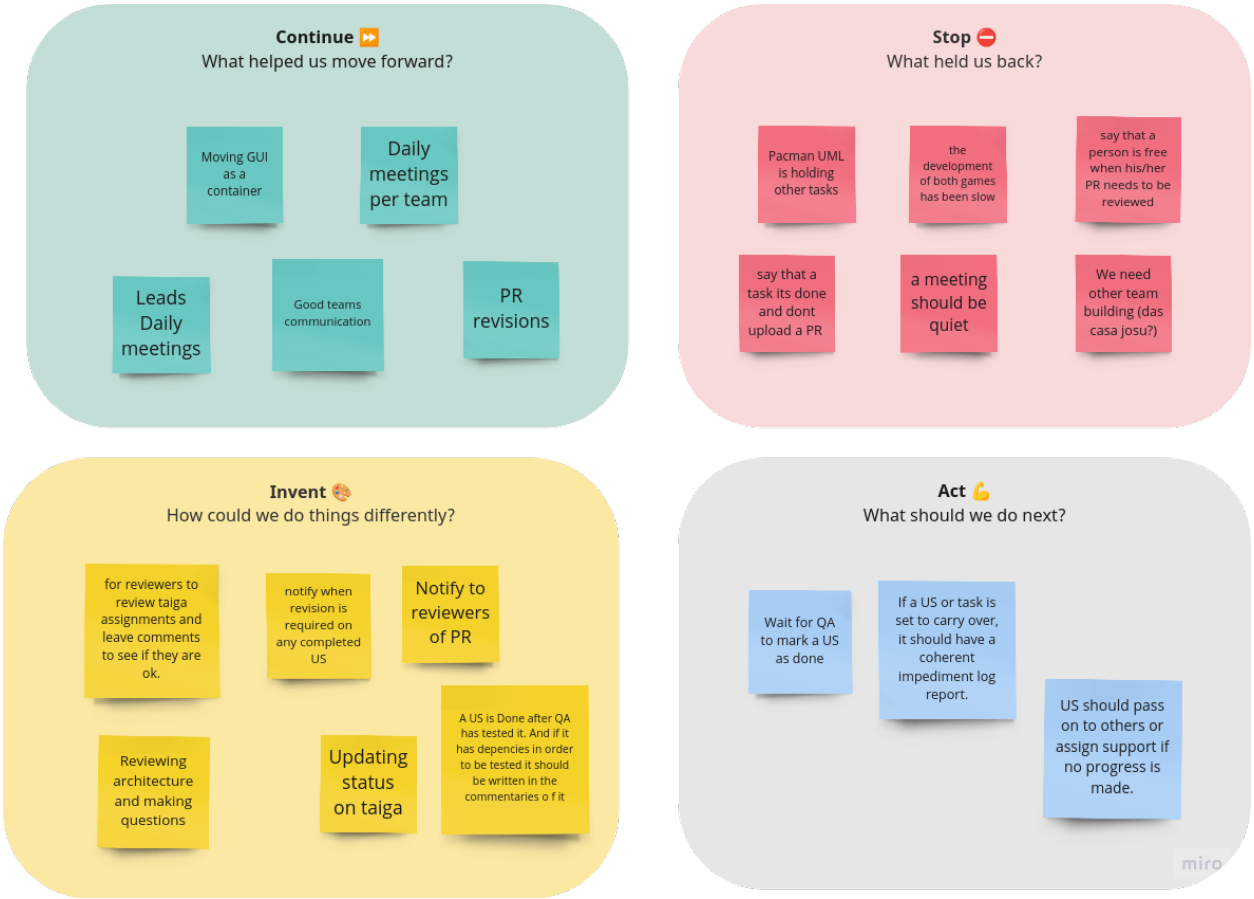
4.2.5 Burn Down Chart

Link: Sprint 3 Board on Taiga.



4.2.6 Start-Stop-Continue-Action Items

Link: Start-Stop-Continue-Action Items Sprint 3 on Miro.



4.2.7 Backlog - Sprint 3

Task Board

Sprint 3 Task Board

Completed User Stories

1. **Update Handler**
   Assigned to: Santiago Concha, Fabian Romero
   Status: Done in Sprint
   Component: BRIDGE

2. **QA - Test Cases for Games**
   Assigned to: Jose Prado, Luis Espinoza
   Status: Done in Sprint
   Component: QA

Carryover User Stories

1. **US-Input Management**
   Assigned to: Axel Ayala, Luiggy Mamani
   Status: Carryover
   Component: CONSOLE

2. **US-Sound Management**
   Assigned to: Alex Paca
   Status: Carryover
   Component: CONSOLE

3. **US-Input Handler for the Bridge**
   Assigned to: Axel Ayala, Luiggy Mamani
   Status: Carryover
   Component: CONSOLE

Total Points Burned

17

4.2.8 Impediments

For the impediments we registered on the following link:

impediments

4.2.9 Conclusions

For sprint 3 our goal was to implement in the project the different parts of the architecture to communicate with each other in the future, but this goal could not be fully achieved as some parts could not be implemented within the sprint duration.

4.2.10 Action items

- US should pass on to others or assign support if no progress is made.
- If a US or task is set to carry over, it should have a coherent impediment log report.

4.2.11 Epics

For Sprint 3 we have the following epics:

- Input Management
- Sound management
- Input handler for the bridge
- Update Handler

- UML Pacman Game
- Render handler implementation

5 Practical Framework

5.1 Sprint 1

5.1.1 Introduction

In this sprint, our focus was on starting the contract for the developer and keep investigation about execution for the game using the console we define. That's why we started with UMLs for our first game (Tic Tac Toe), console and our contract (frontend-library), that is the Bridge. At the same, we had been working on some Spikes for the console, such as to define a graphic library for it, communication via JNI, apply shared memory between a game and console and finally, register input events, like mouse and keyboard. All these Skipes were related in the context of the technologies used in the project.

5.1.2 UMLs

We had been starting on the UMLs for the contract, Tic Tac Toe game and console.

- Task ID - 17: Console Diagram on Taiga (Comments Section)
- Task ID - 18: Library UML Diagram on GitHub
- Task ID - 22: Tic Tac Toe UML Diagram on GitHub

5.1.3 Spikes

As we mentioned before, there were some Spikes to work with and here is the documentation generated by them:

- Task ID - 19: Graphic Libraries Documentation on Google Docs
- Task ID - 20: JNI Documentation on GitHub
- Task ID - 21: Shared Memory Documentation on GitHub
- Task ID - 24: Input Listener Documentation on Word

5.1.4 POCs

In this Sprint, we had no Proofs of Concept (POCs).

5.1.5 Technical Justification

During Sprint 1, the team exclusively focused on designing UMLs and generating Spikes for console. This decision was based on the following reasons:

5.1.6 High - Level Architecture

The architecture followed in this sprint was based on the initial architecture.

- Architecture: Architecture Initial Version You can see it on:
- C4 DSL: Visualizer

5.1.7 Epics

For this Sprint 1, we have the following epics:

- Bridge
- Console
- Graphics
- Games

5.1.8 Sprint Backlog

Completed Spikes and User Stories.

- Task ID - 19: Spike Graphics Libraries for Console
- Task ID - 20: Spike Console with JNI
- Task ID - 21: Spike Shared Memory between Game - Console
- Task ID - 24: Spike Mouse and Keyboard Events Listener
- Task ID - 18: UML Library
- Task ID - 22: UML Tic Tac Toe Game
- Task ID - 17: UML Console
- Task ID - 30: Frontend Library Implementation
- Task ID - 35: Tic Tac Toe Mockups and Sprites

5.1.9 Conclusions

In Sprint 1, our focus was give that initial step in the game and the console, also of reviewing some documentation for our spikes. The most important goals achieved were complete the UML for the contract and console too, besides of defining the graphic library, working with JNI, register input events and apply shared memory between the game and console.

5.2 Sprint 5

5.2.1 Introduction

In this sprint, our focus was on completing the communication between the console and the game using JNI. Backend tasks were assigned for realizing user stories related to establishing communication between the game and the console, while frontend tasks included various implementations for the Pacman game.

5.2.2 Spikes

No spikes were conducted in this sprint.

5.2.3 POCs

No proofs of concept (POCs) were conducted in this sprint.

5.2.4 Technical Justification

During Sprint 0, the team exclusively focused on conducting spikes instead of developing POCs. This decision was based on the following reasons:

5.2.5 High-Level Architecture (HLArchitecture)

The architecture followed in this sprint was based on JNI communication between game, bridge, console, and screen.

- Architecture: Link to architecture
- C4 DSL Link: Visualizer

5.2.6 Product Backlog

Done Completed spikes.

- UML Pacman Game: US id-72
- Execution Script: US id-176
- Pacman Character: US id-79
- Pacman Game Score: US id-84
- Pacman Map: US id-82

Carry Overs

- Tic Tac Toe Game: US id-25
- Bridge Screen Communication: US id-135
- Collisions on Pacman: US id-124
- Pacman Game Initialization: US id-110
- Game Loop Conditions: US id-166
- Pacman Collectables: US id-166

5.3 Impediments

For the impediments we develop a word file:

impediments

5.3.1 Conclusions

In Sprint 5, our focus was on achieving communication between the game and the console, but the goal was not achieved in this sprint. The most significant progress was seen in the realm of games, specifically in the development of the Pacman Game.

5.3.2 action items

- Add priority as tags in the US to make it more visible

5.3.3 Epics

For Sprint 5, we have the following epics:

- Screen
- Bridge

- Console
- Pacman Game
- Tic Tac Toe Game

5.4 Sprint 6

5.4.1 Introduction

During sprint 6 the team focused on refactoring the script with the latest architecture, and refactor socket for screen, and finishing the US related to the pacman game.

5.4.2 Spikes

- Refactor script update with latests architecture: US id-215
- Refactor socket client for screen bassed on latest architecture: US id-216

5.4.3 POCs

No proofs of concept (POCs) were conducted in this sprint.

5.4.4 Technical Justification

In Sprint 6 the decision was made to refactor the script update and priorize US of pacman. The decision was made based on the following:

- Being able to execute the game.
- Terminate the US of pacman that were in carry over in order to finish pacman game.

5.4.5 Product Backlog

Done Completed spikes.

- Refactor script update with latests architecture: US id-215
- Game communication: US id-130
- US: bridge screen communication: US id-135
- Game loop conditions: US id-166
- Colisions on Pacman Game: US id-124
- Pacman Collectables: US id-80
- Pacman Game Refactor: US id-236

Carry Overs

- Refactor socket client for screen bassed on latest architecture: US id-216
- Tic Tac Toe Game Implementation: US id-25
- US: Pacman game initialize: US id-110

5.4.6 Impediments

For the impediments we registered on the following link:

impediments

5.4.7 Conclusions

For this sprint our goal was the refactoring of both the script and the socket, at the same time to document all the sprints and finish the US of pacman, the goal was not met in its entirety since it was not possible to finish the entire US of pacman and the socket refactor.

5.4.8 action items

- Fast and concrete code reviews.
- Self management at assigning US or tasks.
- Improve our US creation in product backlog.

5.4.9 Epics

For Sprint 6, we have the following epics:

- Refactor script

- Refactor socket
- Pacman Game
- Tic Tac Toe Game

5.5 Architecture changes

5.5.1 Introduction

software project plays a pivotal role in determining its success and sustainability. A well-defined architecture not only lays the foundation for the development process but also ensures that the system is scalable, maintainable, and adaptable to future requirements.

5.5.2 Architecture Evolution

We will be reviewing the different changes that the architecture has had throughout the development of the project, which were two in total, we will delve into more details below.

Basic Architecture First version of our architecture, these are the first concepts we had to be able to synthesize the idea of communication between console and video game based on an approach with LLVM given the capabilities of the tool such as having a run-time, the ability to go from a high-level language to a low level.

- First Architecture concept: Architecture

Architecture based on JNI Based on the spikes made in sprint 0 regarding LLVM and a JNI alternative, it was seen that although it was possible to use LLVM for our purposes, the tool was not made for communication between languages, LLVM is a set of tools focused on creating of compilers, although it had a tool developed for what we wanted, it was deprecated and abandoned, so LLVM was declared unviable, JNI as the alternative and based on the results of the JNI spike that was successful and showed a continuous communication between Java and Rust, it was decided to opt for an approach with JNI.

- Second Architecture concept: Architecture
- Spike JVM: JNI

Architecture based on JNI moving GUI to a container It was observed that the communication that would be formed between console and bridge in terms of GUI was not clear. So for the game to run properly, the GUI should be running along with the game; and the current approach ties the graphical execution to the console, where each routine would have to create a GUI instance in order to fulfill its needs.

- Architecture based on JNI concept: Architecture

5.5.3 Architecture based on Socket communication IPC

For this last change there is a merge on bridge with console, using console as a middle-man between bridge and screen meant an extra layer of control, but the price is the complexity of implementation and maintainability. One point to change the architechture was complexity, an independent socket server running alongside the

game and screen to enable their communication. This means console and socket server would need an agreement regarding frames.

- Last Architecture concept: Architecture