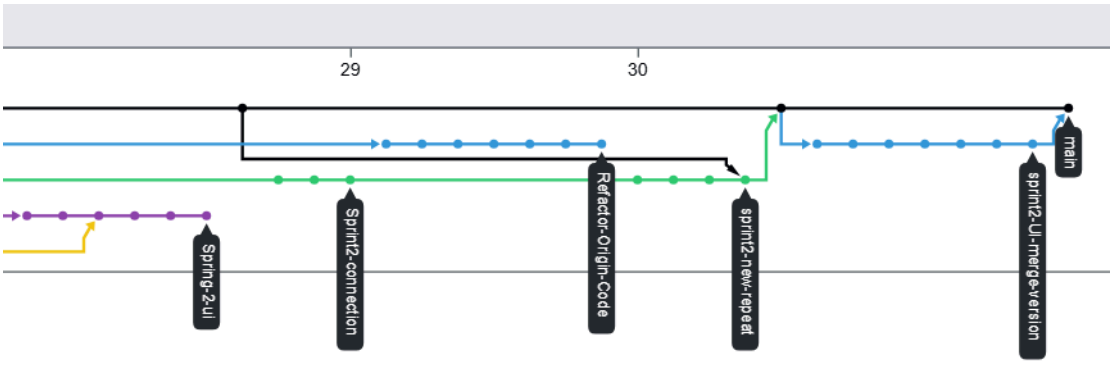
Project Development Process Documentation

Branching Strategy

Our project uses a task-based branching strategy. Each team member creates their own branch based on the part of the task they are working on. After they finish the work, they merge their branch into the main branch.

Branch Structure

- **Main branch: main**
  This is the default and stable branch. Only completed and tested code should be merged here. No one should write code directly in main.
- **Feature branches: named by task or sprint**
  Each developer creates a branch based on their assigned task. Branch names usually include the sprint number or the function being developed.
  Examples:
    - Sprint2-connection
    - sprint2-new-repeat
    - Spring-2-ui



Branch Workflow

1. Create a branch before starting a task.
2. Develop and test your code in your own branch.
3. Open a Pull Request (PR) to main after you finish your work.
    - Include a clear description of what you changed.
4. Team member reviews the code (see Code Review Process).
5. Merge the branch into main after approval.

Commit Conventions

We use a simple and clear commit format based on a lightweight version of conventional commits. This helps make the project history easier to read and understand.

Format

Each commit message should follow this structure:

<type>: <short summary>

- **Type:** describes the kind of change.
- **Summary**: is a short sentence describing what you did. Use the base form of the verb (e.g., add, fix, update).

Common Types

Type	Purpose	Example
feat	Add a new feature	feat: add sound effect to block
fix	Fix a bug	fix: correct repeat block logic
docs	Update comments or documentation	docs: translate all comments to English
style	Change code style or design (no logic change)	style: adjust button padding
refactor	Improve code structure (no feature/bug)	refactor: simplify UI component loop
test	Add or modify tests	test: add test for sound toggle
chore	Changes to tools or dependencies	chore: update dependency version

Code Review Process

We follow a basic and clear code review process to keep the main branch stable and the code quality high.

Pull Request Required

All changes must be submitted through a Pull Request (PR) before merging into the main branch.
No one should push code directly to main.

Reviewer

Each PR must be reviewed by at least one other team member.
The reviewer is not fixed. You can choose any teammate to review your code.
The reviewer should check the code and give feedback before the PR is approved.

What to Check

The reviewer should make sure the code:

- Works correctly (no bugs)
- Completes the assigned task or feature
- Has clear and readable variable/function names
- Follows the team's coding style
- Does not include unused or repeated code
- Includes helpful comments if needed

Note: We do not require testing as part of the review for now.

Tools

- We use GitHub Pull Requests to do code reviews.
- We currently do not use CI/CD tools for automatic checks.

Project Management Tools and Collaboration Rules

Developers should follow the following steps :

1. Create an issue on GitHub.
2. Locally create a feature branch :
   `git checkout -b {branch name}` like git checkout -b Sprint2 UI merge version for issue #48 (closed).
3. Develop the specified requirements
4. Set a time estimate and task priority, then keep track of the hours spent.
5. Before committing, clean up dead code, duplicated and useless comments.
   - Commit regularly. Each commit should be small, self-contained.
   - Keep your git commits organised - make it tell a story. Follow the team's reading recommendations: Write Better Commits, Build Better Projects
6. Before submitting a merge request, maintain a semi-linear git history with git fetch && git rebase origin/main
7. Create the merge request (MR) title identical to the issue title.
   - link it to the issue by typing `Closes #48` in the description
   - Assign yourself as the Assignee and one of the peers as the Reviewer.

Useful commands

Sync change from main branch :
`git checkout main`
`git pull`

Remember to run this every time before push:
`git fetch`
`git rebase origin/main`

For committing your code:

```
git add path/to/your/file
git commit -m "commit messages
git push
```

For the first push:

```
git push --set-upstream origin {branch name}
```