NTNU
Kunnskap for en bedre verden

TDT4501 - Specialization Project

Developing an educational game for learning Git version control system

Game of Git

Authors:
Steven Francis
Frithjof Thorvik

Codebase available at:
https://gitlab.stud.idi.ntnu.no/gog/git-cooking

December, 2022

Contents

1 Introduction

1.1 Context

This paper will address the processes that took place during the implementation and realization of a project proposed by George Adrian Stoica. The project aimed to design and develop an educational game that can facilitate the learning of core concepts in Git version control system. Our task was to design, implement, and evaluate an application that can be used by students for learning version control. This required an extensive pre-production phase, including research, eliciting requirements, design, and testing different approaches before starting the development of the application.

1.2 Personal motivation

Both of us are quite similar when it comes to the type of games we play, what we like about our field of study, and our experiences when it comes to development.

Even though neither of us have much experience when it comes to game development or any specific goals in becoming game developers, the concept of developing an educational game spiked an interest in us both. Especially when the game would teach Git to beginners, something we could relate to, and feel would be beneficial to a lot of people. With that in mind, we wanted to create something that we would be proud of sharing with others.

1.3 Goals

The goal of this project was to create a functional prototype for an educational game teaching Git to students who are new to Git and wants to learn it in a more efficient and fun way.

To achieve this goal we want to do research with the goal to: *Investigate which principles of game based learning and game design that makes learning engaging, and map these to creating a educational game for learning Git version control system.*

In addition, our goal was to research existing solutions, interview the target group, and gather enough data to make informed decisions about the directions to take the concept for our game.

2 Prestudy

The goal for the pre-study was to investigate which principles of game based learning and game design make learning engaging, and map these to creating a educational game for learning Git version control system. This goal was broken down into five questions:

- What are the core concepts of the Git version control system?
- What are important principles to consider when creating an educational game?
- What can game design say about these principles?
- What is the process for making a game?
- Are there existing games for learning Git and which elements of educational games do they employ?

This section of the report will try to answer these questions.

2.1 Git

Git is a free distributed version control system. A version control system is a system responsible for tracking changes to computer programs, documents or other collections of information, so that specific versions can be retrieved later. With a distributed system, every clone of a server acts as a backup of the data, making it easy to restore data if the server is corrupted (Chacon & Straub 2022).

Figure 1 shows the main work flow of Git. You can checkout a specific version of your files from the Git repository to the working directory, where you can modify the files. After these modifications are made, they can be added to the staging area, which stores information about the modified file in its current state. Modifying the file further results in changes in the working directory, but the staging area is not affected, unless you add those new changes again. When you commit, the changes in the staging area are saved in the Git repository. As a result, the Git repository acts as a database that stores all the project files, along with their revision history. Continuing we will give an explanation for some of the core concepts in Git.

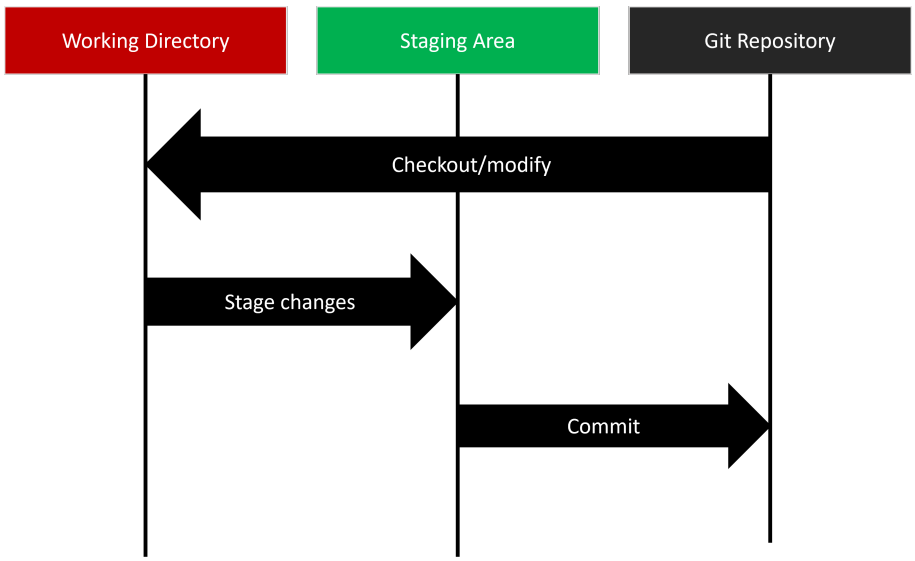


Figure 1: The main work flow of Git with working directory, staging area and Git repository.

Working Directory: The working directory acts as the platform where you can modify files and make changes to your project. It can also be viewed as a checkout of a version/snapshot of your

repository, where you can make changes. These changes are not saved in Git before you commit them or stage them. Within the working directory, files can be in one of two states, tracked or untracked. Tracked files are files that existed in the previous snapshot or in the staging area, whilst untracked files are new files not yet tracked in the Git repository (Chacon & Straub 2022).

Staging Area: A modified file from the working directory can be added to the staging area. The staging area contains information about which changes will be stored in the next commit. Meaning that the staging area contains information about the modified files in the state it was when it was added to the staging area (Chacon & Straub 2022).

Commit: A commit takes all the changes added to the staging area and saves them to the Git repository. Git stores its data in a series of snapshots, so whenever you commit, you save a snapshot of the current state of your files. For efficiency, if a file has not changed, Git links to the previous file, instead of storing a duplicate. Every commit contains a pointer to its previous commit, resulting in a stream of snapshots as seen in Figure 2 (Chacon & Straub 2022).

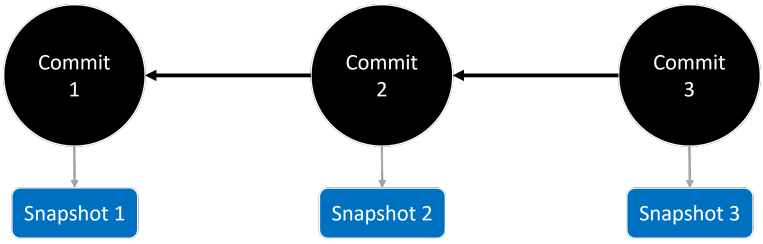


Figure 2: In Git commits point to its parent commit and to a snapshot. This results in the data of Git being represented as a stream of snapshots.

Branch: A branch in Git is essentially a pointer that points to a commit. The default branch is called the main branch or master branch. Whenever you commit, this branch points to the last commit you made, and for each commit, the main branch pointer automatically moves forward. Whenever you make a new branch you create a new pointer that points to the commit you branched from (Chacon & Straub 2022). This new branch pointer can move independently from other branches. Git also has a special pointer called "HEAD" which can be used to know which branch you are currently on (see Figure 3). In this way you can switch between branches. This allows you to experiment with changes by making new branches, doing some changes in that branch and switching to another branch to make other changes. For these changes to be integrated with each other, Git supports a feature called merging, which integrates the changes with each other; the next paragraph goes into more depth on merging.

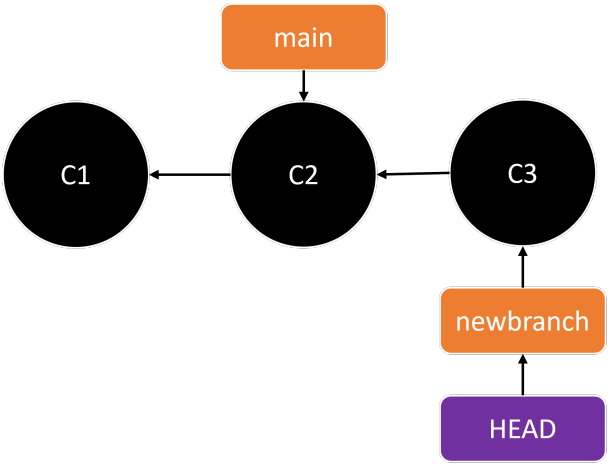
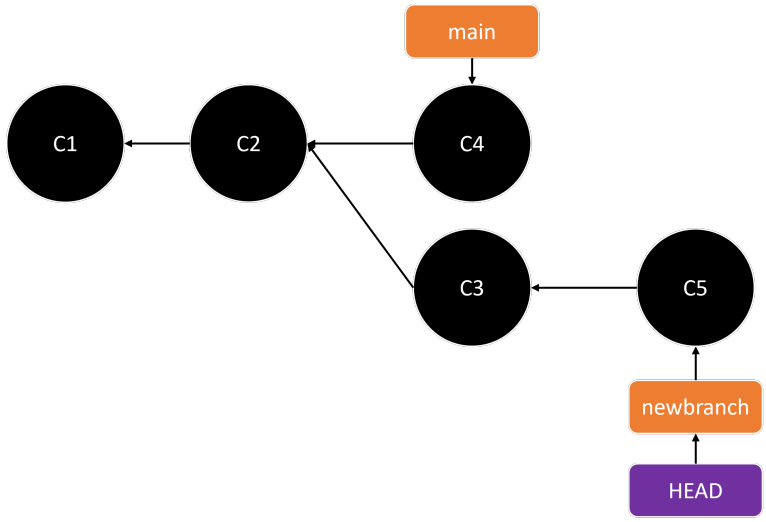


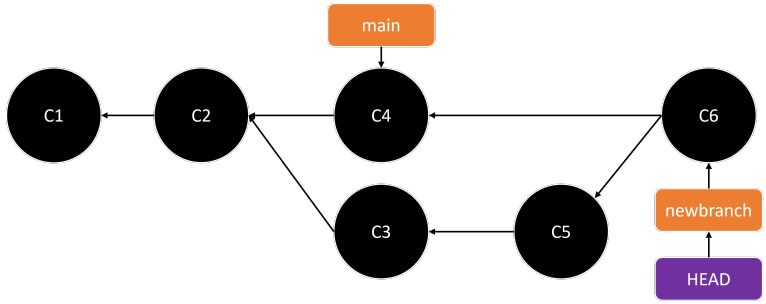
Figure 3: A branch is a pointer to a commit. And the "HEAD" pointer can point to a branch, keeping track of the active branch.

Merge: Merging is how Git manages to integrate changes from different branches together. Based on the changes made, the merge can happen in two different ways: fast-forward merge or three-way

merge. A fast-forward merge occurs whenever the changes you are trying to merge can be reached by following the commit history. When this happens, the merge can be simplified by moving the pointer forward, as there are no conflicting changes to merge together (Chacon & Straub 2022). This is what would happen if *main* was merged with *newbranch* in Figure 3. A three-way merge occurs when the commit history has diverged down the line, see Figure 4 (a). This happens when the commit of the branch you are trying to merge is not a direct ancestor of the branch you are merging in. In this case, Git creates a new commit, called a merge-commit, shown in Figure 4 (b), which takes the changes from the two branches and combines them. This commit is special compared to a normal commit as it has more than one parent (Chacon & Straub 2022). In the case that the changes between the two branches are in conflict, the three-way merge cannot proceed. This is called a merge conflict and it happens when the same part of a file is changed in different ways in the two branches you want to merge. When this happens, Git pauses the merge and cannot proceed until you have resolved the conflicting changes. Once the conflicting changes have been resolved, the merge can proceed in the same way as before.



(a) Divergent branches.



(b) A merge commit.

Figure 4: A three-way merge.

Remote: Git enables you to collaborate on Git projects through remote repositories. A remote repository is similar to a normal Git repository, except that it is hosted on the Internet. This remote repository can then be accessed by multiple people, allowing collaboration through pushing and pulling in Git. Pushing and pulling is the act of sending data to the remote (pushing) and getting data from the remote (pulling). This enables you to work on a directory make some changes, commit those changes and the push those changes to the remote. The person you are collaborating with can then pull those changes, gaining access to the changes you made. If you want to start working on an existing remote repository, you can clone the project, which pulls down every file and its history from the remote project (Chacon & Straub 2022).

Git Commands: Git commands provide the main way of interacting with Git. These commands are usually executed in the command-line on your computer or in the terminal on your code editor.

Some of the basic commands are:

- `git init`: for creating a new repository
- `git add`: for adding modified files to the staging area
- `git commit -m`: for saving changes in the staging area to the repository
- `git status`: for listing modified files, and files in the staging area
- `git checkout`: for switching branch and checking it into your working directory
- `git branch`: for creating a new branch or listing up existing branches
- `git merge`: for merging the changes of two branches together
- `git clone`: for obtaining an existing remote repository
- `git push`: sending committed changes to a remote repository
- `git fetch`: retrieving branches from a remote repository

2.2 Educational games

The goal of this section is to highlight the most important principles to consider when creating an educational game.

The difference between an educational game and a traditional game is that educational games are more complex. Their design does not require just story, art and software development, in addition educational games need to encompass pedagogical strategies, learning approaches and appropriate feedback (Lameras et al. 2016). Therefore when designing educational games it is especially important to use game features which take into account established learning theories. Multiple articles cite constructivism as a useful learning theory for educational games (Qian & Clark 2016, Ibrahim & Jaafar 2009, Dondlinger 2007, Lameras et al. 2016). The constructivist viewpoint is described by Annetta & Leonard (2010) as the idea that people learn from connecting new experiences to their prior experiences. The main idea is that people learn from actively experimenting with new knowledge and not by passively taking in information. An educational game that incorporates this into the design will have a greater chance of success (Qian & Clark 2016).

A comparative analyses by Tahir & Wang (2019*b*) concludes that, in addition to pedagogy/learning theories, game factors such as mechanics, dynamics, narratives, aesthetics, and goals are essential aspects of designing educational games. The analysis goes on to conclude that to achieve engagement and immersion, the design phase of educational games should place a greater emphasis on linking learning objectives to game objectives in an efficient way. One can also find this conclusion by Shute & Valerie (2011), they conclude that educational games suffer from the fact that the learning objectives disrupt the flow. Therefore, games should be created with learning objectives that are less obtrusive and better linked to game objectives.

An attempt to link learning objectives to game objectives has been made by Lameras et al. (2016) by linking Boom's taxonomy of learning outcomes with game attributes. The findings of this study show that to achieve memorization and recall of information the game should include challenge and repetition. To achieve understanding, problem solving and the ability to analyze concepts, a game should include storytelling, role-playing, game levels, collaboration and game missions. Therefore one should try to include these mechanics when designing educational games. This conclusion is backed up by multiple literature reviews which also mention collaboration, challenge, role-playing and story telling as often used game mechanics in educational games (Qian & Clark 2016, Avila Pesantez & Rivera 2017, Laine & Lindberg 2020).

Laine & Lindberg (2020) goes into more depth on these game attributes by providing design principles for each attribute. When it comes to challenge, their review recommends providing adjustable difficulty in-order to cater to a wider audience. They also recommend reducing complexity

of challenges, vary the type of challenges and provide adequate time to solve the challenges. Both Lameras et al. (2016) and Laine & Lindberg (2020) recommend repetition of challenges, which allows players to try out different strategies. For collaboration Laine & Lindberg (2020) suggest offering tools like chats or multiplayer activities. Their research also suggest providing opportunities for competition through for example leader-boards and achievements. Providing means for expressing social status and recognition among players, along with providing means to form and mange groups along with allowing players to distinguish themselves in these groups are also good for collaboration. When it comes to story telling Laine & Lindberg (2020) recommends creating a story that players can relate to, in a fantasy context, with role-playing experience and possibilities for the player to interact with the story. Their research also encourages the use of humour in the story as it can benefit learning.

Another important aspect to consider when creating educational games is to provide learning in the correct context, as doing so can provide higher intrinsic motivation compared to traditional teaching (Lameras et al. 2016). The research done by Laine & Lindberg (2020) recommends providing the learning content at the right transparency level, which they suggest to be in a realistic context for higher education and more subtly for young children. In addition the content should be relevant from two perspectives: both in regards to the target context (where the game is played) and in regards to the chosen game genre and mechanics. Their research also suggests grounding the game in a real-world context with familiar activities, which is supported by Qian & Clark (2016) and Chorianopoulos & Giannakos (2014) who both say that it is important to use game design elements that are already proven successfully in the gaming industry, and that will be familiar to the user.

Lastly, an important feature of game based learning is its potential to increase students' motivation for learning (Qian & Clark 2016). Besides getting students to spend more time on learning, motivation has positive effects on memory and personal development (Dondlinger 2007). Other sources also mentions that motivational factors are central when designing educational games (Laine & Lindberg 2020, Avila Pesantez & Rivera 2017).

2.3 Game design

The previous section highlighted certain game mechanics that are useful when designing educational games. While this is a good starting point, it does not help us with implementing these game mechanics in a game. The goal of this section is to figure out if literature on game design can help us with this. Lazzaro (2008) provides literature on how to design games for unlocking emotions in video games. According to Lazzaro people play games not for the graphics or genre but in order to feel specific emotions, and these emotions come from different game mechanics. Lazzaro categories these different game mechanics into four quadrants called The Four Fun Keys.

Challenge: One of the Four Fun keys is the Hard Fun key and it comes from game mechanics related to challenge and breaking goals into small achievable steps. The Hard Fun key outlines that in order for challenge to be rewarding for players the difficulty must be balanced with the player skills. Games that are too easy becomes boring and games that are too hard become too frustrating. The goal is to provide a challenge that lies in the middle between too easy and too hard, keeping the player in the zone (Lazzaro 2008). The challenge also needs to be varied and gradually increasing to keep the player interest. Not starting off too difficulty but inviting players to experiment with different strategies and to develop mastery. The pace of new challenges should therefore be balanced to keep the tension in the game without overwhelming the player and make him frustrated (Sweetser & Wyeth 2005). In addition, for a game to be challenging it needs to be related to a goal with uncertain attainment. However the goal should be obvious and compelling, with enough feedback to tell the players how they are progressing towards the goal (Malone 1980). Malone also highlights the need for goals on several levels. One can achieve this by including meta goals above the main goals. An example is to make it a goal to achieve the main goal more efficiently. Goals can also be broken down into sub goals that must be completed to achieve the main goal. There is also a need for attaining goals to be rewarding, or the challenge is not worth persisting. Here it is important to make the reward of the goal clear before a player achieves it, so that the player is motivated to persist against the challenge. This reward can be giving something valuable to the player, or if the challenge is at the right level, then overcoming the challenge might

be a reward in itself (Schell 2008).

Collaboration: People play games to hang out with their friends even though they might not even like playing games. This is the principle for the People Fun key (Lazzaro 2008). Playing together generates positive emotions, feelings of trust and companionship. People fun is primarily enjoyed by players because they feel closer to their friends after laughing with them. Lazzaro suggest offering choices for interaction between people and opportunities to cooperate, express themselves and personalize to increase People Fun. Another aspect that attracts people to play online games is the opportunity to join virtual communities, chatting with friends and gradually improving their playable character (Sweetser & Wyeth 2005). This is also apparent in the Socialiser player type proposed by Bartle (1996). The Socialiser player type see the game as the scene where things happen to other players, their goal is to get to know people and understand them and to form lasting relationships. Therefore when designing a game one should consider adding channels for communicating and for interacting with other players.

Roleplay and storytelling: Immersion is in an important part of game design. This feeling of deep involvement in the game make players become less aware of themselves and their surroundings which is a enjoyable for them. The goal of a game is therefore to make players forget they are playing a game (Sweetser & Wyeth 2005). The Easy Fun key is one of Lazzaro's Four Fun keys and it relates to mechanics that inspire curiosity, with the end goal of unlocking emotions like awe and wonder. Mechanics like role-playing and story-telling that create an interesting fantasy or inspire imagination in the player contribute to Easy Fun (Lazzaro 2008). A good narrative is important for immersion into a game. People want to know who their characters are and what is happening to feel a part of the story (Sweetser & Wyeth 2005). For the narrative of a game to be engaging it should be novel and surprising but not incomprehensible (Malone 1980). If the game becomes too predictable the player quits out of boredom and if it is too novel the player quits because the game doesn't make sense. A game should therefore strive to keep a balance between familiarity and novelty (Lazzaro 2008).

These previous mechanics have been related to mechanics that are deemed important for learning. However there are other game mechanics that are useful in general when designing games. Some of these are control, concentration and feedback, which are all important aspects of the GameFlow model that evaluate player enjoyment in games (Sweetser & Wyeth 2005).

Control: Players must be able to feel a sense of control over their actions. They should feel like their intentions are translate to in-game behaviours and be able to feel a sense of mastery of the control system (Sweetser & Wyeth 2005). The controls should be suitable for the players and the context, sensitive and accurate, consistent and familiar (Laine & Lindberg 2020).

Concentration: A game should require the entire concentration of the player, without being burdened by tasks that don't feel important. Distractions from the main game should be minimized by reducing nongame-related interactions. And the main game should captivate the players attention by providing something worth attending to (Sweetser & Wyeth 2005).

Feedback: Games require immediate and appropriate feedback to let the players identify their progress towards the goals of the game (Sweetser & Wyeth 2005). It is important to provide instructions and tutorials that let the player know how the game is played (Laine & Lindberg 2020). The feedback of the game should be presented in a way that minimizes the possibility of self-esteem damage (Malone 1980).

2.4 Game development

The purpose of this section is to understand the process of making a game.

The first step in creating a game is to come up with an idea for what you want your game to be. This is a creative process with a focus on non-functional requirements such as enjoyment, aesthetics and fun. This first step of game development is usually captured in a game design document, which is not necessarily a formal document. The goal of a game design document is to capture the creative vision for the video game, everything from concept statements, story,

characters, gameplay, and aesthetics might be included here (Callele et al. 2005). When it comes to this phase of development, Schell (2008) the author of *The Art of Game Design* provides a guide for the creative process of making a game:

1. State the problem.
2. Brainstorm some possible solutions.
3. Choose a solution.
4. List the risks of using that solution.
5. Build prototypes to mitigate the risks.
6. Test the prototypes. If they are good enough, stop.
7. State the new problems you are trying to solve, and go to step 2.

The creation of the game design document can be considered as pre-production in the game development process, and the production phase is associated with writing the code for the game. Schell (2008) says that the transition from pre-production to production happens once you have two completely finished levels. Before that you are still figuring out the fundamental design of your game. This transition from pre-production to production is important, as most failures occur in this transition (Callele et al. 2005). Callele et al. (2005) highlights that one of the reasons for these failures is that software requirements for the emotional aspects of video games are not captured officially. Callele's research also highlights that errors can stem from including software requirements in the design document or by eliciting requirements from the design document in an ad-hoc basis. He goes on to emphasize that *two* sets of documentation must be created and maintained; a design document to capture the creative vision and a requirement document for the software artifact. A successful project includes feedback and feed-forward between these two documents, as requirements are always emergent, a significant aspect of the creative design process. From this research it is apparent that one must focus on managing the transition from pre-production to production.

Once in the production phase of game development one is required to think about how to structure the software development to best capture the creative design of the game. One difference between game development and other software development is that game development also requires managing assets, animation, sound and dialog. Wang & Nordmark (2015) found that software architecture is an important part of the game development process, especially for managing the complexity of the software. And the software architecture need to be modifiable to support new gameplay mechanics that might be discovered whilst the game is being made.

2.5 Existing solutions

To get a better understanding of what it means to create a game for learning Git we looked at existing games that try achieve the same thing. The games we found were Github Minesweeper[1], git-game[2], githug[3], Twilio[4], OhMyGit![5] and LearnGitBranching[6]. In this section we present and analyze some of these solutions. We will analyze these games using the LEAGUÊ analysis instrument created by Tahir & Wang (2019*a*). The LEAGUÊ analysis instrument consist of three parts for analyzing game-based learning: a primary analysis form, a secondary form to reinforce in-depth analysis, and a reflection form. For brevity we will only apply the primary analysis form, which poses a set of questions for the six dimensions in the LEAGUÊ framerwork (Learning, Environment, Affective Reactions, Game Factors, Usability and User). These questions can help

[1] https://profy.dev/project/github-minesweeper
[2] https://github.com/git-game/git-game
[3] https://github.com/Gazler/githug
[4] https://www.twilio.com/quest/learn/open-source
[5] https://ohmygit.org/
[6] https://learngitbranching.js.org/

for analyzing learning games and highlight its weaknesses and strengths (Tahir & Wang 2019*a*). The questions are shown in Figure 27 in Appendix A, but for brevity an answer to each question will not be provided in this report. Instead, the goal of this section is to get an overview of the strengths and weaknesses of existing solutions.

2.5.1 LearnGitBranching

LearnGitBranching is a game/visualization tool for learning Git. The game features a node-based representation of Git commits, where the goal is to execute the right Git commands to get the correct node-structure. Each level consist of a goal to reach, which is an image of a node-structure as shown on the right side of the screen in Figure 5. Each level starts of by giving you a tutorial on the Git commands that are relevant for the current level (see Figure 6). In the game you can navigate the node-structure by using Git commands, as one would navigate Git normally; in addition the game offers a more interactive feature where you can navigate by clicking the nodes in the game. The node-structure update dynamically as you execute Git commands, allowing you to visualize what you are doing. Once you have gotten the node-structure to equal the goal, you are prompted with a screen which shows you have many commands you used and how many commands their solution used.

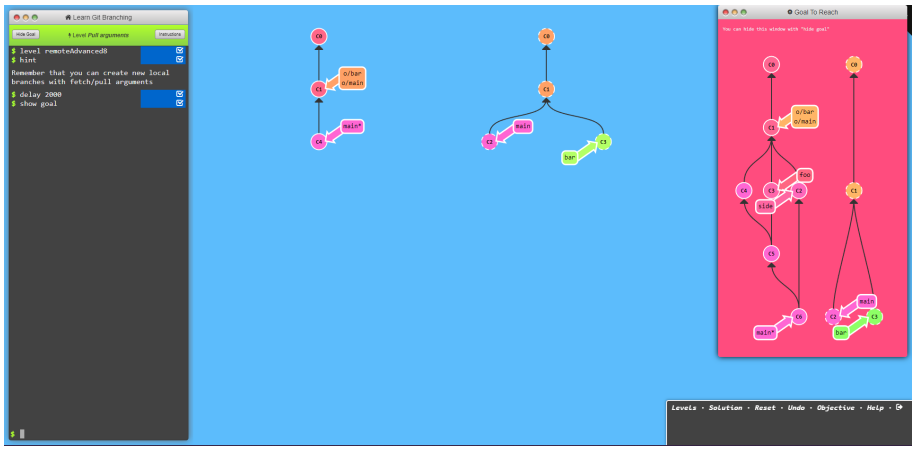


Figure 5: The left side of the screen is used to write and execute commands. The middle of the screen show your current node-structure. The right side show the target node-structure.

Strengths: The strength of this game is its colourful visualization of Git commits. This is paired with intuitive and simple controls, which makes the experience interactive for the player. This interactive experience makes it easy for the player to experiment and test out their own strategies for solving each puzzle. The game also offers a thorough tutorial before each level, which makes it accessible for everyone looking to learn. The goals of the game is provided on two different levels: a overarching goal of completing every level, plus a meta-goal of completing the level in the correct amount of steps. In this way the player is encouraged to replay levels for even more learning.

Weaknesses: Besides the bright and intuitive interface, the game doesn't offer much to engage the user. There is no story-line, no characters and little reward for continuing to play. Because of this there is a risk of players becoming bored and disinterested in continuing playing. The difficulty of the game is too easy, where most of the levels requires you to do exactly as the tutorial just taught you. There is a lack of creative problem solving and encouragement for trying different strategies. Because of this, the game becomes a place to get an introduction to Git and not a place to hone your Git skills.

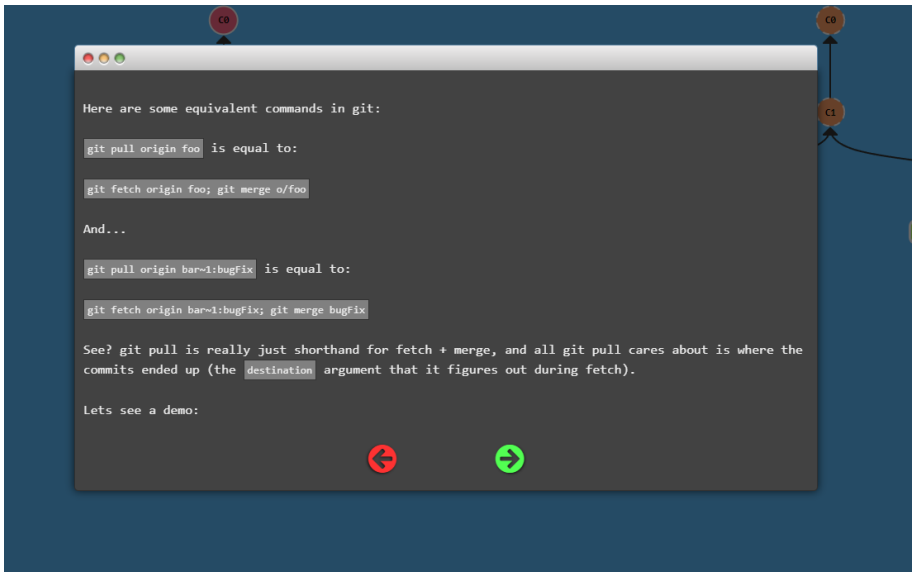


Figure 6: Example of a tutorial screen in LearnGitBranching.

2.5.2 Twilio

Twilio is a game for learning how to contribute to open source projects on GitHub. The game features an open world which you navigate with a customizable playable character as shown in Figure 7. In this world you follow a story-line where you solve challenges using Git. The game is not only played within the game world, but one must also write real code and contribute to a real open source project in GitHub. An example of a challenge you must solve in the game is shown in Figure 8.

Figure 7: The world of Twilio.

Strengths: The strength of this game is the focus on immersion and storytelling. The game has beautiful graphics set in a 2D world, which makes you forget that you are playing a game to learn Git. The story-line is engaging and can motivate the player to continue playing. In addition the game provides a customizable playable character which allows the player to become more immersed in the game. The game also provides a quest-line where the player is required to do things that are not directly related to Git. These quest-lines also provide rewards including new items for the

character and experience points. This adds multiple layers to the game, where the player can play both to learn Git and also because they feel engaged in the quest-line. The game also provides tutorials for each challenge related to Git, so that even beginners can learn in the game.

Weaknesses: The biggest weakness of the game is that the challenges are too easy. Most of the challenges only requires you to execute certain steps in GitHub, without any catering towards creative problem solving. In addition, the game is relatively short and once the game is completed there is no replay value. Another drawback is the emptiness of the game world. Even though the game is engaging at first sight, the player quickly realizes that there isn't much depth to the world and the player might loose motivation to continue exploring.

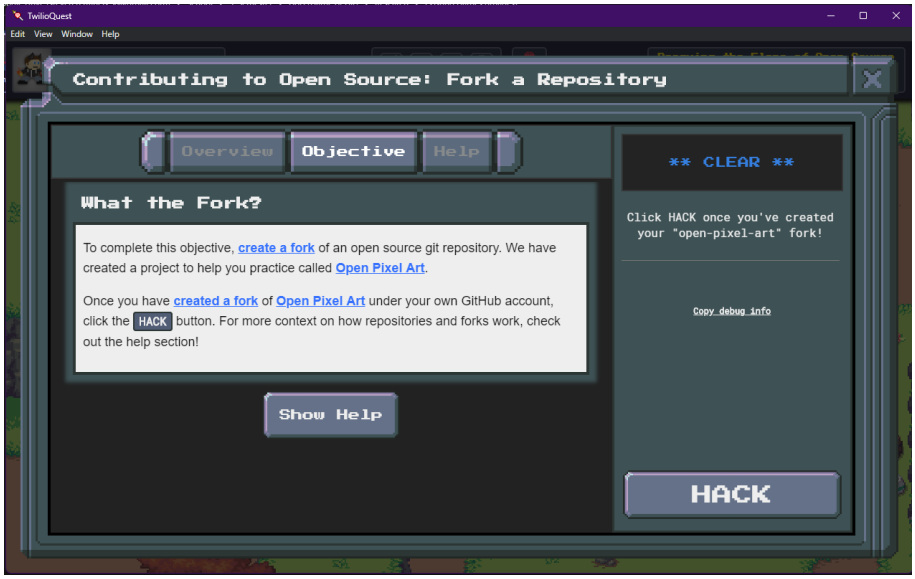


Figure 8: An example of a challenge in Twilio, where you are required to fork an open source repository in GitHub.

2.5.3 OhMyGit!

OhMyGit! visualizes Git repositories' internal structures in real time, meaning that players can see the results of their actions immediately. The game features a playing card interface to help beginners memorize newly introduced Git commands. In addition the game provides a terminal where you can execute Git commands in the usual way. A snippet of the game's interface is shown in Figure 9. The game is level based, where each level provides an unique Git challenge. The challenges are presented within a story-context, where the goal is to solve the problem using Git commands and by manipulating the repository's files.

Strengths: The greatest strength of this game is its wide coverage of Git concepts. Compared to the two previous games, this games provides the most varied competence in Git concepts. Another strength of this game is the use of story-line which makes the objectives of the game more engaging an motivating. The game does not provide Git tutorials, but instead allows the player to use Git cards instead of writing commands manually. This makes it easier for beginners who find it difficult to remember what each command do. The gradual increase in difficulty, helps ease the player into Git concepts. Another great aspect of the game is that it provides a context for Git by imagining it as a time machine. This makes it both engaging and easier to understand how Git works.

Weaknesses: The biggest drawback of the game is that some of the objectives are unclear. The player can easily become stuck because it can be difficult to understand what the game wants you to do. Similarly to the previous two games, this game does not encourage creative problem solving and there is usually only one way to solve each puzzle. This limits learning because the player cannot try out his own strategy and understanding of Git. A small drawback of the game is related

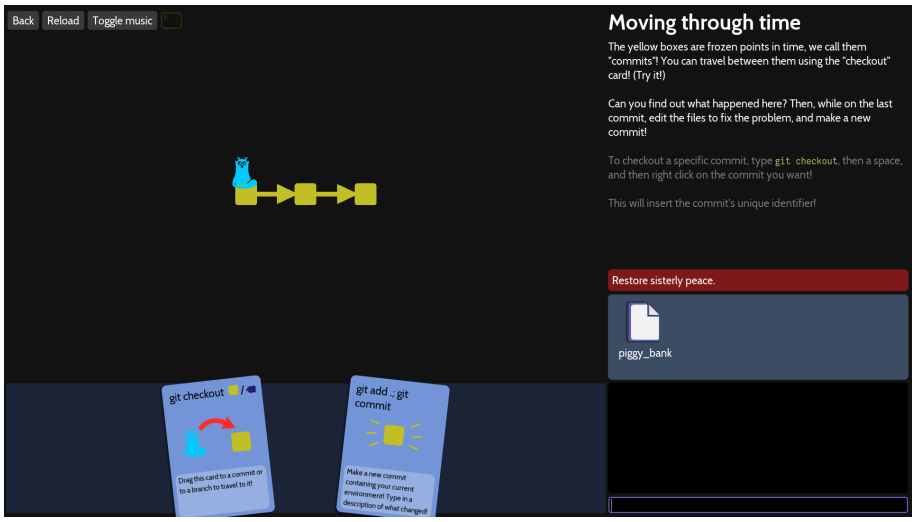


Figure 9: The interface for OhMyGit!, showing the Git playing cards, a visualization of the Git repository, a terminal and a story objective.

to performance issues, as the game can sometimes be a bit laggy. This is a problem as it becomes an irritation moment when playing and will take the player out of flow.

3 Design & Prototyping

The goal of this section is to design and prototype a pedagogical game with the goal of facilitating learning of Git version control system. The design tries to use the research found in Section 2 to best achieve this goal. One important aspect that was found in the pre-study was the need to focus on linking learning objectives with game objectives (Tahir & Wang 2019*b*). Because of this a big part of the design process consisted of conceptually testing out different game approaches, to try to figure out which approach would support the objective of learning core concept in Git version control system. Other parts of the design process consisted of semi-structured interviews with students to get a better understanding of what they want to learn in Git and what they find difficult to learn in Git. The last step in the design process was to create visual prototypes to get feedback from students.

In the early part of the design process we followed principles from Design Thinking as taught in the course *TMM4220 - Innovation by Design Thinking*[7] at NTNU. This course focuses on needs-finding though empathy, rapid prototyping and user-testing, with the goal of designing a product that is useful for the target group. This process, with rapid prototyping, aligns well with the creative process of making a game by Schell (2008) presented in the pre-study.

3.1 Interviews

To get a better understanding of which concepts of Git students found the most difficult, we conducted semi-structured interviews with six students, some experienced with Git and some less experienced with Git. These interviews focused on empathy to figure out the needs of students when learning Git. The interview was guided by questions such as: *"What is your relationship with Git?"*, *"What do you find difficult to understand about Git?"*, *"What do you think is easy to understand about Git?"* and *"What do you want to learn in Git?"*.

From these interviews we learned that students want to know what kind of features exist in Git and what those features do. They said that it is hard to know what to do in Git when you do not

[7] https://www.ntnu.edu/studies/courses/TMM4220#tab=omEmnet

know what you can do, and they would like to get an introduction to the features Git provides. The more experienced students reported that they found it difficult to learn Git, and that they felt like they were never taught how to use it. They said that they were just thrown into using Git, without any guidance or knowledge on best practises. They would therefore enjoy playing a game that would help them learn best practises in Git. Some of the students also mentioned that they would like an arena to test out their Git skills, as they find it intimidating to practice Git in the real world. They said that they found it difficult to practice Git without a safe environment to do so, and that they needed an environment to just experiment with different Git features. Another student mentioned that they found it hard to understand how all the parts of Git worked together, and they wanted to learn the abstracts of what happens when you work in Git.

3.2 Protoyping

After the interviews we decided to create three conceptual prototypes which focuses on different needs found in the interviews. Prototype 1 focuses on the need to get an overview of best practises in Git. Prototype 2 focuses on the need to understand how all the parts of Git worked together on an abstract level. Prototype 3 is a mix between the abstract and the realistic, and it focused on providing an arena for experimenting with Git, whilst also giving an overview of which Git commands that exists and what they do. The prototypes are mostly conceptual, with a visual representation created in Figma[8] to make it easier for students to give feedback.

3.2.1 Prototype 1

The first prototype is a game that is more technically oriented, where you solve real problems you might encounter in Git, but it has different game elements that make it fun to play. In the game you start as an novice programmer and get various quests/scenarios that you have to solve with Git in order to level up and get through the game. See Figure 10 for the visual representation.

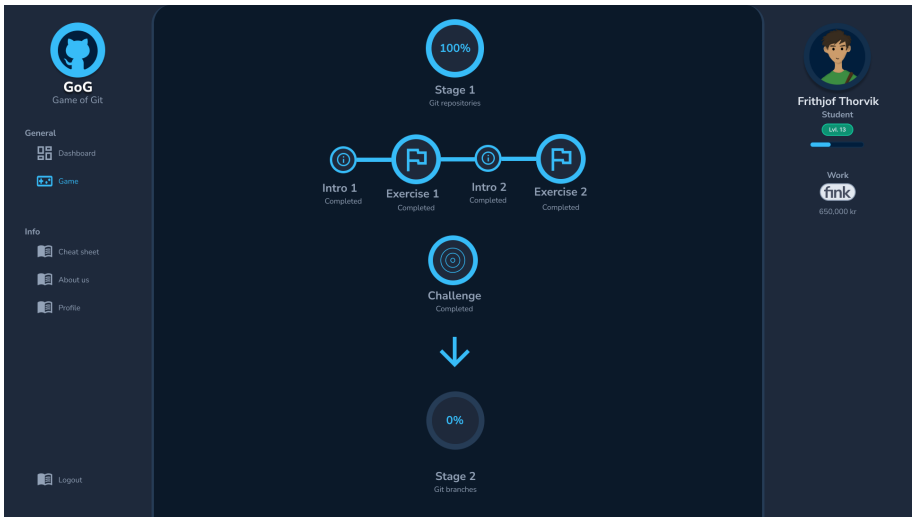


Figure 10: Prototype 1 with focus on best practice and realistic scenarios.

The game's story will work its way through various concepts in Git, from the most basic processes such as connecting to a repository, staging, committing, and pushing changes, to somewhat more advanced concepts such as fixing various problems where you have to remove previous commits, merge conflicts, etc. The quests/scenarios will be text-based, and are carried out in the initial phase through multiple choice, until eventually solving the problems in the terminal. The goal of this game is aimed more at solving tasks that are related to real situations.

[8] https://www.figma.com/

The research this prototype is based on is Qian & Clark (2016), who says for university students the game should be in a real-world context. It also is based in the need for challenge, role-playing and storytelling as mentioned in Section 2.

3.2.2 Prototype 2

The second prototype is a game that is not technically oriented, but it is based on the abstract concept of Git. The goal of this game is to connect Git concepts to the game when you're later introduced to Git or have to learn it. The visual representation for the game is shown in Figure 11.

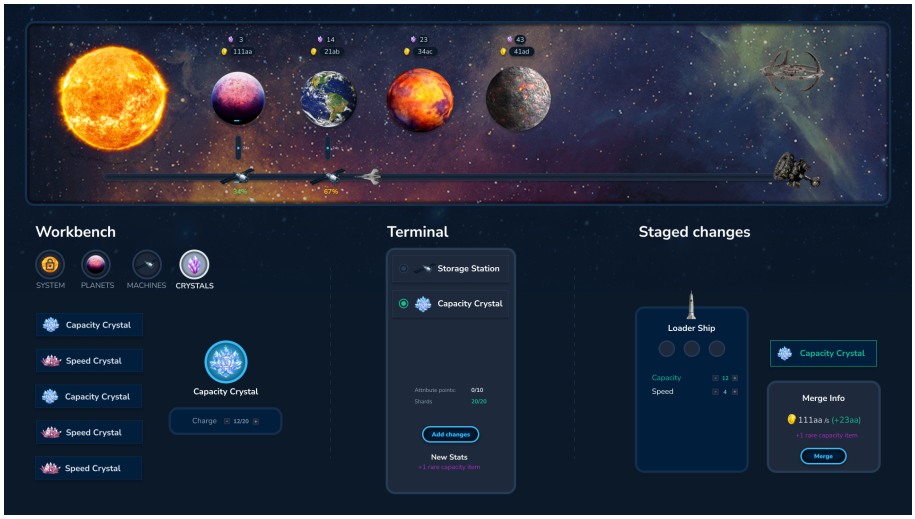


Figure 11: Prototype 2 with focus on understanding the abstracts of Git.

The game is a simulation where you get money and materials by mining planets. You have various spaceships that pick up and deliver the resources, and you can level them up and make changes to the ships to optimize the process. When you make changes, you can test between different changes on the different spaceships, and try to find the most optimal change. You can then choose to keep these changes. In this game you don't learn Git commands, but rather the goal is for you to gain an understanding of important Git concepts seen in a context that is not directly Git, but reminiscent of Git.

The research this prototype is based on is Tahir & Wang (2019*b*) and Shute & Valerie (2011), where the goal is to make the learning objectives less obtrusive and disruptive. The goal of this prototype was to see if people were interested in a game where the learning happens subconsciously; with focus on more flow and less distraction from learning objectives.

3.2.3 Prototype 3

The third prototype involves using Git to solve problems, but the problems are not directly related to reality. This game involves intrinsic fantasy where you have to use Git commands to perform actions within the fantasy context. The main goal is to be able to experiment with Git commands and Git processes in a more interactive and fun way than in a real project. The visual representation of the game is shown in Figure 12.

The concept of the game is a restaurant/cooking game that involves serving the right food to customers. The game is set in the future where you work as a work from home chef. You put together dishes in a terminal reminiscent of code editors, where you change files to create the dishes. You can use Git commands to stage and commit the dishes so that they are served to the customer. The goal is for you to get to know various Git commands directly, while also having fun serving the right food to the customer. The goal is also to make the process of repeating various commands more interactive so that it is easier to remember the various commands.

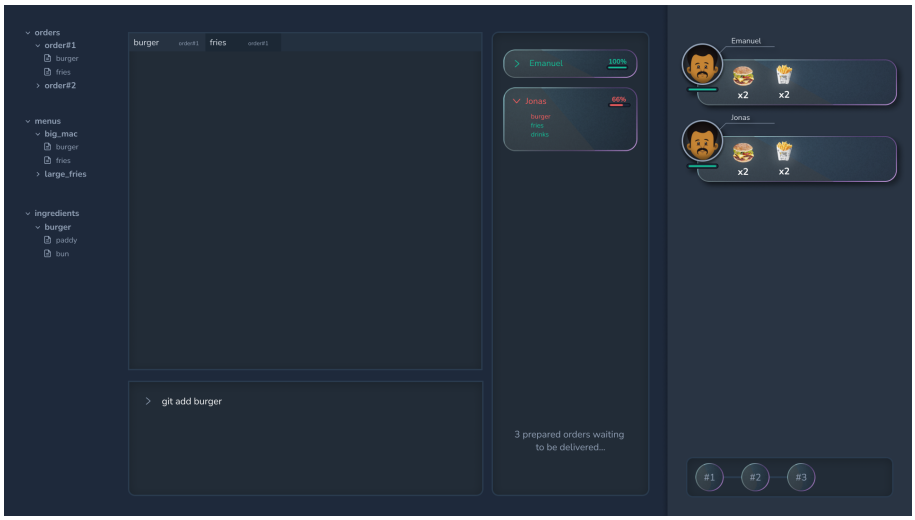


Figure 12: Prototype 3 with focus on providing an arena to experiment with Git.

This prototype is based on the research by Chorianopoulos & Giannakos (2014) and Shute & Valerie (2011) that say to use game design elements that are already proven successfully in the gaming industry. Therefore this prototype uses many design elements from the popular genre of cooking games. It also uses concepts like repetition of challenges, clear goals and storytelling as mentioned in Section 2.

3.2.4 Prototype feedback

The three conceptual prototypes were explained to five students, along with showing the visual representations. From this we received valuable feedback for which prototype to develop further. For prototype 1 the students said that they like that the learning objective is clear, and that you can choose what to learn. However, they found the idea to not be as engaging as the other prototypes; where one student said that they have tried similar games, but would become bored and quit before finishing. For prototype 2 the students said that it looked both fun and engaging. However the students pointed out that they felt discouraged since the learning objectives weren't as clear. Here one student asked what the point of playing is if you don't know if you will learn anything. Another student said that the game was too abstract and that the whole point for learning Git is to learn the commands. Prototype 3 received the most positive feedback from the students. They found the game concept intriguing and fun, and they said that they wanted to play this game the most. One student mentioned that they had played cooking games before and wanted to try out this game as well. There was also constructive criticism for this prototype, where students said that the game looked difficult to understand and needed a tutorial. In addition, some said that it was important to support advanced Git commands, as this is what they wanted to learn the most.

Based on this feedback, we decided to move on with prototype 3, developing the idea further, and eventually creating a game based on this idea. In addition to the feedback received, we based our decisions on the research discussed in Section 2. From this we found that prototype 1 lacked engaging gameplay that would motivate the player to learn. The learning objectives were also not well linked with the game objectives, where the fantasy of the story together with real life scenarios would potentially break the immersion and engagement of players. Prototype 2 didn't provide learning at the right transparency level; it was too abstract. On the other hand, Prototype 3 has a better transparency level, mixing both abstraction through fantasy and real learning through the use of Git commands. In addition, the game has more potential compared to prototype 1 for motivating and engaging players. Lastly, prototype 2 is based on familiar game concepts already proven successful, which was recommended by Chorianopoulos & Giannakos (2014) and Qian & Clark (2016).

4 Solution

The solution is a further developed version of prototype 3, created in Section 3. The solution focused on feedback received from students (see Section 3.2.4) and from the research discussed in Section 2. The solution especially focused on: linking learning objectives with game objectives, providing challenge, providing immersion through storytelling, familiar gameplay, and good feedback.

Linking learning objectives with game objectives: To achieve this, the solution focuses on learning Git through gameplay, without disturbing the gameplay with learning objectives. Instead, the learning objectives are infused into the gameplay, by making the player play the game using Git commands.

Challenge: The solution focuses on providing challenges that are not too difficult, with gradually increasing difficulty as the game progresses. The challenges in the solution focus on creative problem solving, where the player can try out their own strategies. In addition, the progression of the game allows for repetition of challenges. The solution also focuses on providing clear goals at different levels, with sufficient rewards for completing them.

Storytelling: The solution focuses on immersing the gameplay in a fantasy context; With a focus on intrinsic fantasy, making the gameplay and learning objectives more engaging and immersive.

Familiar gameplay: The solution borrows gameplay mechanics from the cooking game genre. This includes a day-based progression, unlockable upgrades, matching orders for customers, and receiving in-game currency based on performance.

Feedback: The solution provides feedback to the player for progression toward different goals. In addition, the solution provides instructions and tutorials to guide the player along the way. This was done both because of feedback from students (see Section 3.2.4) and based on research on game design (see Section 2)

4.1 Game concept

Based on the conclusions made during the pre-development phase, the game would merge Git concepts into the world of cooking games, resulting in the name: *Git Cooking*.

Git Cooking is placed in a world where restaurants have digitized and streamlined many of their processes. Food and orders are created with machines controlled digitally by the restaurant's staff. These machines are built around Git to create and provide customers' orders. Upon entering the game, the player is a new employee in one of these restaurants and will have to learn how to use this system to help the owner grow their restaurant and business.

During the day, when the restaurant is open, the player will have to fulfill the orders placed throughout the day. This process requires the player to create the orders and register these changes in the terminal with the proper Git commands, for the machines to create the orders. After each day, the player will receive a certain amount of money depending on the amount and quality of fulfilled orders. At the end of each day, the player can purchase different upgrades to generate more revenue and unlock new features to increase productivity within the game.

4.2 Gameplay

The game starts on Day 1, where your job is to fulfill orders provided by customers. Before each day starts the player is required to fetch orders from the remote repository. This will provide the player with three remote branches to choose from, which provide different orders and rewards. The player starts the game by checking out a remote branch using the Git command `git checkout [branch name]`. Once this is done the player is taken to the work screen where new customers arrive throughout the day, keeping you busy with new orders to fulfill. The goal of the game is

to create items that match the customers order, add those item to the staging area, and then committing these items. Once the items are committed they are delivered to the customer. At the end of the day, the player is required to push his changes back to the remote repository. If the player has worked on multiple branches he has to remember to push all the branches back up to the remote. After this is done, the player can choose to end day, which takes him to the summary screen. Here the player receives in-game currency based on how close the items for each branch match the orders. The player can then use the in-game currency to buy upgrades, unlock new ingredients and buy new Git commands, before another day starts and the cycle continues. This game loop allows for repetition of challenges which was found to be important for game based learning in Section 2.

4.2.1 Fetch screen

The fetch screen, shown in Figure 13, is an important part of the game, as it is where the player is required to use git commands to access orders from the available remote branches. In order to do this, the player must first use the `git fetch` command to retrieve the list of branches that are available. Once this is done, the player will be shown all the available options and some details for each option, which can help the player make a decision on which branch to work on. By using the `git checkout` command followed by the name of the branch they wish to access in order to start working on that branch.

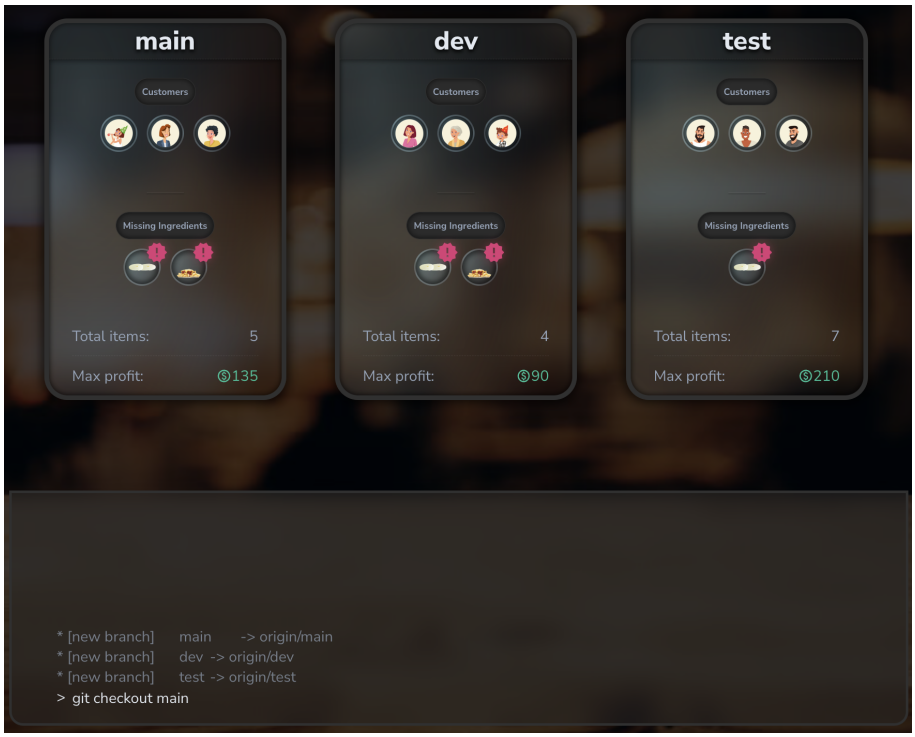


Figure 13: The fetch screen showing the available branches that can be used, in addition to missing ingredients and how many orders that will show up.

4.2.2 Work screen

After checking out a remote branch the player is taken to the work screen, shown in Figure 14. The work screen is the place where the player get orders from customers, create items and then delivering those items to the customer. This part of the gameplay consists of five visual elements: the file structure on the left side of the screen, the item editor in the center of the screen, the terminal interface beneath the editor, the staging area to the right of the editor, and the order section on the right side of the screen. The visual elements are designed to look similar to a code editor, making it familiar to the player, and helping to set the context for learning Git.

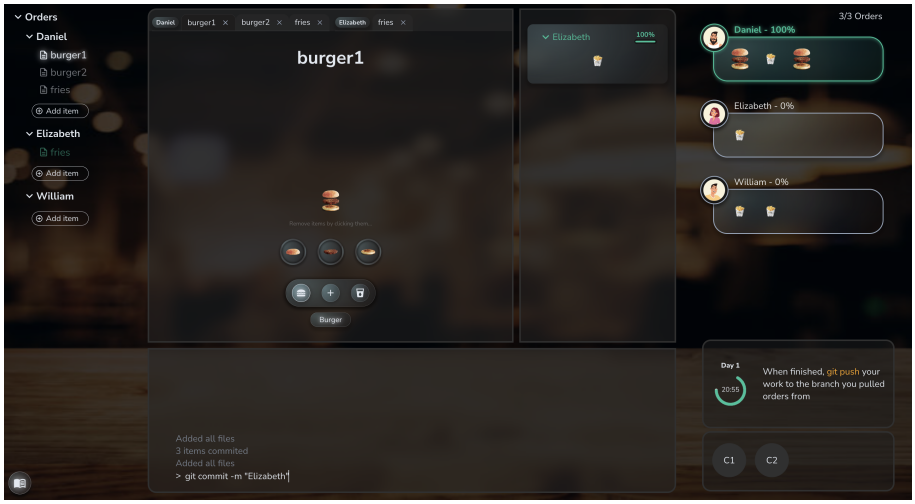


Figure 14: The work screen is the place where most of the game play takes place.

Folders and items: To fulfill an order you start by creating a folder in the file structure, under *"Orders"*, on the left side of the screen. For each order that arrives, an opportunity to create the folder is presented in the file structure. The player can then press the button with the name of the order to create a folder for the order. Within the folder, the player is free to add items with the *"Add item"* button. An item corresponds with an item pictured in the order from the customer, and it is the job of the player to use the item editor to match each item with the items in the order. The text for an item provides visual feedback to the player, based on if the item is modified, staged, or not (see Figure 15). Orange text indicates that the item has been modified, green text indicates it has been staged, and white text indicates that the item has not been modified or staged.

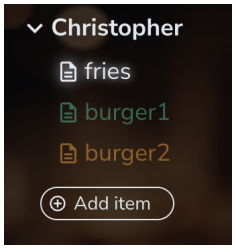


Figure 15: An item can be in three different states, orange indicates a modified item, green a staged item, and white indicates an item that has not been modified or staged

Item Editor: The item editor, in the center of the screen, provides a way for the player to modify an item. Here the player has to choose an item type to create, which presents the player with different ingredients to create the item with. The goal for the player is to add the ingredients in the correct order, to match the item requested by the customer. The item editor is shown in Figure 16.

Figure 16: The item editor provides a way for the player to modify items.

Staging area: When an item has been added to the staging area the player will get visual feedback on his progress towards matching his order with the order provided by the customer, as shown in Figure 17. In the staging area, the player will see the items he has staged, along wit the order it belongs to. Here, the player also has the opportunity to see what the items he is staging looks like. These features provide the player with clear feedback on his progress towards the goal of meeting the demands of the customer, which is important for game design and learning (see Section 2).

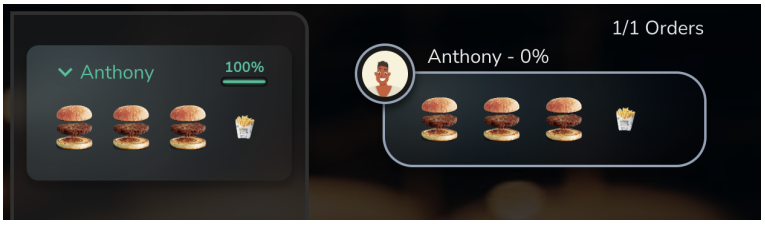


Figure 17: This image show that once an order with the correct items has been staged, the player will see a percentage indicating how close the staged items match the customer's order.

Orders: As the day progresses, customer orders will appear on the right side of the screen, see Figure 18. An order consist of different items that the player must match to satisfy the customer. To the right of the name of the customer the player can see the percentage of how well his committed items match that of the customer. Orders arrive evenly throughout the day, but in-order to keep the attention of the player, a new order will arrive after another order has been completed. This game mechanic provide a clear goal for the player, with a challenge that is not too difficult or complex and it keeps the attention of the player, which is based on research described in Section 2.

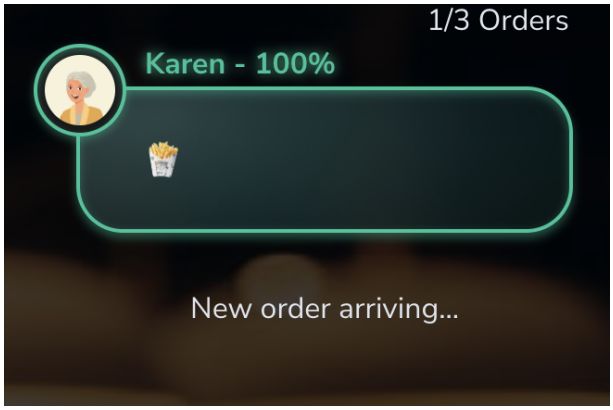


Figure 18: New orders arrive throughout the day, and when you have completed a previous order.

Terminal interface: The terminal interface is the player's main way of interacting with the game by using Git commands. To add items to the staging area, the player must use the git command `git add .` or `git add [path]`, to add all items or a specific item to the staging area. The player can then commit these items using the command `git commit -m [message]` with a specified commit message. The goal is for the player to become familiar with both the git commands and the concepts of git by using these commands to play the game. The game also support the `git restore .` or `git restore [path]` and `git restore --staged .` or `git restore --staged [path]` commands to be able to restore modified item and staged items. All these commands work conceptually similar to Git, the only difference is that instead of working with files, within the game you modify, add, commit and restore items. See Section 4.3 for all the Git commands supported by the game.

Another feature of the terminal is that it provides feedback to the user as he executes the command, as shown in Figure 19.

Figure 19: The terminal is the users main way of interacting with the game. The terminal also provide feedback when executing commands.

Infobox: In the lower right corner the player can see an info box which provides the player with feedback on how long is left of the day, and on which day he is. Relevant hints will also be displayed here, as shown in Figure 20. This mechanic is implemented to increase the guidance provided to the player within in the game, as recommended by the research in Section 2.

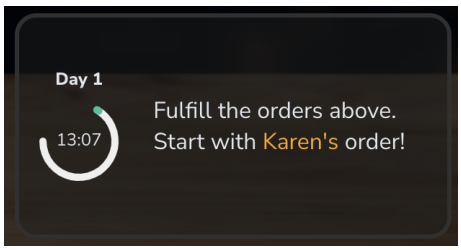


Figure 20: The info box show the time of day, as well as useful hints for the player.

Commit history: Under the info box, in the lower right corner, the player can see the commit history, which provides a visualization for the commits he has made (see Figure 21).

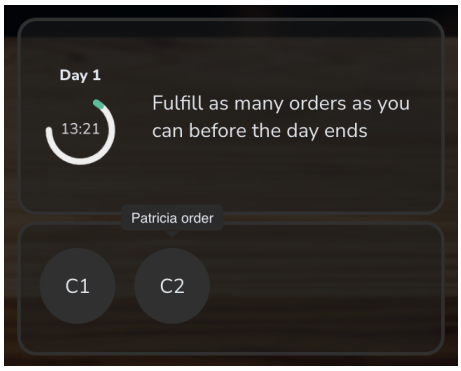


Figure 21: The commit history shows all commits, in addition to their commit message when hovered.

4.2.3 Summary screen & Store screen

After a day is completed the player is presented with a summary screen, which gives an overview of how the player did that day, see Figure 22. Based on how well the player did, he receives in-game currency, which he can spend in the store screen, shown in Figure 23. This game mechanic is based on the research in Section 2, which recommends to give the player goals on multiple levels. This is done by providing the player more in-game currency if he finishes the day early and if he gets 100% on every order. Which comes in addition to the main goal of completing the day.

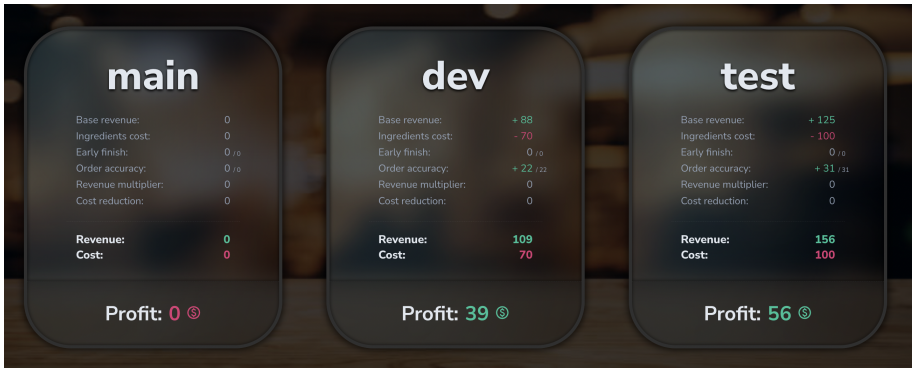


Figure 22: The summary screen provides feedback to the player for how well the day went. Here the player will see how much profit was gained in the different branches.

Within the store screen the player can navigate between three categories:

Upgrades: In the *Upgrades* category the player can buy upgrades that increase the revenue he can gain at the end of each day, and upgrades that increase the length of the day. Some of the upgrades are unlocked only once the player has reached a certain day, and they can be bought with in-game currency. These kinds of upgrades are common within the cooking game genre and provides a goal for the player to obtain all the upgrades.

Git Commands: In the *Git Commands* category the player can unlock new Git commands. These Git commands allow the player quality of life upgrades that will make navigating the gameplay easier. By incorporating unlockable Git commands into the upgrade screen, the player can gradually learn new commands at their own pace. This mechanic was implement based on the research in Section 2 which mentions to provide challenge and learning at a suitable pace.

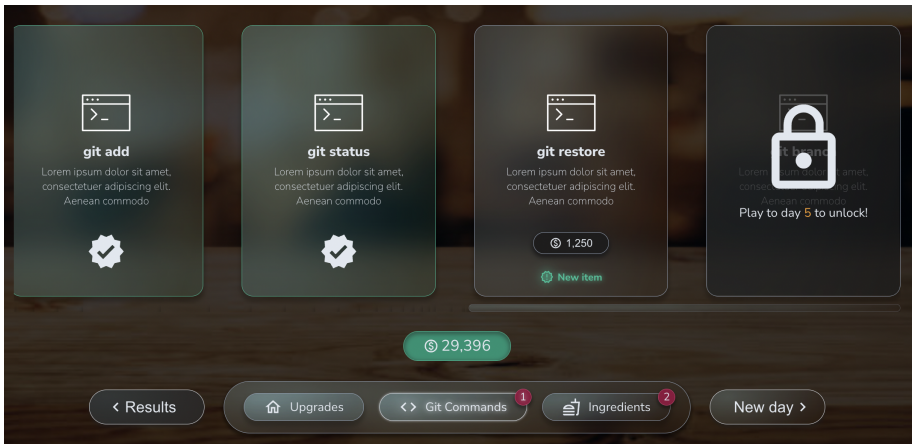


Figure 23: The store is where the player can buy upgrades, git commands and new ingredients.

Ingredients: In the *Ingredients* category the player can buy new ingredients too use. Similar to the *Upgrades* category, the ingredients are bought with in-game currency and some are locked until a certain day is reached. After an ingredient is unlocked there will be a chance for this ingredient to show up in an order moving forward. This will make new orders more diverse, increase the challenge as there are more items to manage, but as a reward the revenue at the end of the day will increase because of these new ingredients. This gameplay mechanic is common within the cooking game genre and provide a goal for the player to unlock all the ingredients, whilst also providing a increase in challenge for the player and more diversity in the gameplay. It is also based on the research from Section 2 which explain that challenge should be varied and increasingly difficult.

4.2.4 Tutorial

The game provides a tutorial that is prompted when the player encounters gameplay mechanics for the first time (see Figure 24). This provides the player with context and story for the game and also acts as a guide for playing the game. In addition, a button for accessing all the tutorials is placed on every game-screen, making it easy for the player to relearn mechanics if he forgets them (see Figure 25). This game mechanic is based on the research in Section 2 which suggest providing adequate instructions and tutorials for playing the game.

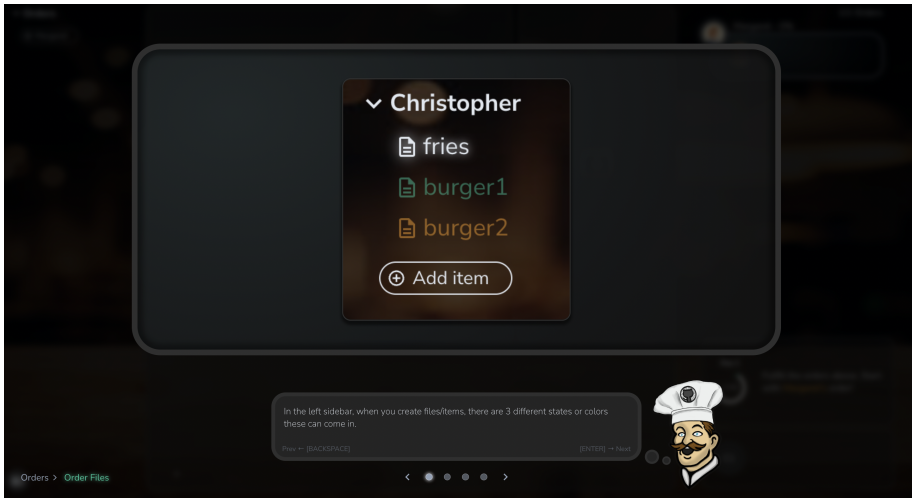


Figure 24: The tutorial screen guides the player throughout the game.

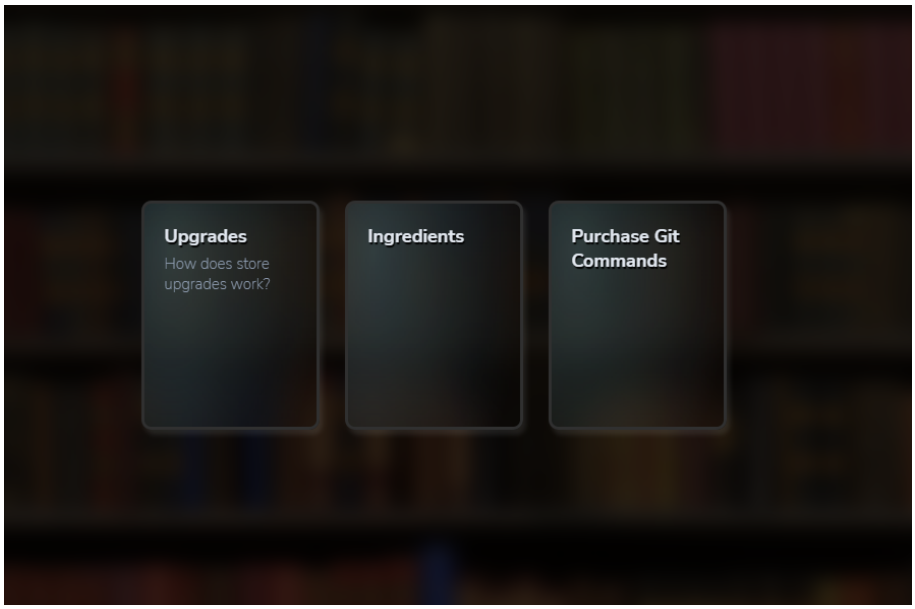


Figure 25: The help screen allows players to revisit previous tutorials.

4.3 Linking Git and Gameplay

An important aspect of the gameplay and the game itself, is how the game objectives are linked to Git. This is essential, as the game's main goal is to teach Git. This section will provide an overview of how different gameplay elements are designed to mimic Git in order to provide learning.

Working Directory, Staging and Committing: To teach the concept of a working directory the game provides a way to add new items and modify those items. However, these changes are not yet stored in Git, and similarly, orders are not completed unless the correct changes are committed. This will teach the difference between a working directory, where you make changes, and a git repository, where the files are saved. The game also provides a staging directory in a similar way to Git. Before modified files can be committed, they have to be added to the staging area, and when you commit, the items in the staging area are saved. The goal of these game mechanics is to teach the flow of modifying, staging and then committing changes in Git. In addition the player is required to learn the correct Git commands and what they do.

Branch: Branches are supported in the game, where the player can checkout both different remote branches and local branches. This works conceptually similarly to Git, where "checking out" a branch will update the working directory to the changes made in the "checked out" branch. The player will learn how to use git commands, such as `git branch` and `git checkout`, to switch between different branches in their repository. They will also understand that committing changes in one branch does not affect the other branches, allowing them to experiment and make changes without affecting the main code-base. Currently, the game does not provide a way of merging and creating new branches, but this could be introduced to the game as further work.

Remote: The game tries to teach the concepts of remote repositories by providing a way for the player to fetch and push to remote branches. At the start of each day the player fetch orders from the remote, where each set of orders are conceptually similar to remote branches in Git. Once these orders are fetched, they can be "checked out" in the same way you can in Git, which creates a new branch locally and set this branch to track the remote branch. At the end off each day the player has to push all the changes he has made to the remote. The player will only gain profit for the changes he has pushed to the remote, which will teach the player that there is a difference between changes made locally and changes stored in the remote repository. Within the game the player has to learn the correct Git commands for fetching and pushing, as well as learning the concept behind what happens when you fetch and push in Git.

Terminal: The game uses a terminal to teach players the most important way to interact with Git. The terminal in the game functions similarly to a terminal in a code editor, and it is the primary means for players to interact with the game. Through the terminal, the player can learn Git commands that are necessary to achieve goals within the game.

4.3.1 Supported commands

This is a list of the supported Git commands in the game and the similarity to Git.

- `git add`: Within the game this command work similarly to Git by adding modified items from the working directory to the staging area. The game supports two variants `git add .` or `git add [path]`, which works as in Git by adding all modified items or the specified item.
- `git commit -m [message]`: Similarly to Git, changes in the staging area are committed to the git repository with this command. In the game this is reflected by updating the percentage completed for each order once a commit is made, to reflect the items saved in the commit.
- `git status`: This commands provides information about modified and staged files, as-well as the currently active branch (similar to Git).
- `git checkout [name]`: In the same way as in Git, this command can be used to checkout remote or local branches. However in the game, the command does not support checking out commits.
- `git branch`: As in Git, this command list up the local branches. The game also support the version `git branch -r`, which list up remote branches.

- `git restore`: This command restore modified items to match the last commit or the last staged version of the item, which is similar to the functionality in Git. The game supports two variants `git restore .` or `git restore [path]`, which works as in Git by restoring all modified items or the specified item.
- `git restore --staged`: This command works as in Git and removes items from the staging area. The game supports two variants `git restore --staged .` or `git restore --staged [path]`, which works as in Git by removing all staged items or the specified item.
- `git fetch`: Similar to Git this command fetches remote branches.
- `git push`: Similar to Git this command pushes changes in the currently active branch to the remote tracking branches. The game supports an additional version `git push origin [branch name]` to specify which branch to push.

4.4 Planning / Architecture / Technology

4.4.1 Technology

When it came to choosing the technology stack for our game, we decided to wait until the main game concepts were decided before making any decisions. The reason for this was to make it easier to choose technologies that would support efficient development. We evaluated well-known game engines like Unity, Unreal Engine, and Godot, as well as widely-used web development tools like React. We looked at the performance, compatibility, and cost of each technology, as well as their alignment with our game's concept and goals. We also considered our familiarity and expertise with each technology.

Unity, Unreal Engine, and Godot are game engines that provide developers with a range of tools and features for creating video games. Unity and Unreal Engine are widely used for developing high-quality, complex games across a variety of platforms. They offer advanced graphics rendering, physics simulation, and scripting capabilities, as well as support for multiple languages and a large community of users and developers. Godot, on the other hand, is a more lightweight game engine that is focused on simplicity and flexibility. It has a user-friendly interface and a robust set of tools, but may not be as powerful or feature-rich as Unity or Unreal Engine. React, on the other hand, is a powerful, versatile technology that allows developers to create interactive and dynamic user interfaces. It is well-suited for creating web applications that do not require the advanced features and capabilities of a game engine like Unity or Unreal Engine.

In terms of game engines, we were aware that Unity, Unreal Engine, and Godot are popular, powerful tools for creating high-quality video games. However, we had very little experience with these engines and were not sure if they would be the best fit for our game concept. On the other hand, we had extensive experience with React and web development in general. In the end, we decided not to use a game engine for our game because it did not require advanced graphics, physics simulation, or scripting. Our game was relatively simple and could be implemented and represented through a web application. Since we had a lot of experience with React and knew that the core mechanics of the game could be realized using this technology, we were confident that we could create the game quickly and efficiently. Using React allowed us to leverage our existing knowledge and skills, and avoid the time and effort required to learn and use a more complex game engine. Overall, our decision to use React was based on our game's requirements, our team's expertise, and our desire to realize a prototype of acceptable quality with minimal effort and cost.

4.4.2 Planning/Process

In this project, we had to come up with a concept for a game that would teach players about Git, and then create a testable prototype of that game. Working in pairs, it was important for us to figure out a plan for how to approach the project and which processes to follow. This involved

doing research on existing solutions, figuring out the concept for our game, developing a prototype, and documenting our process and choices along the way.

The planning phase of the project was crucial for its success. We started by researching existing solutions and looking at what had been done before in order to gain a better understanding of the field. From there, we worked on figuring out the concept for our game and developing a prototype that we could test and iterate on. We also made sure to document our process and choices, so that we could refer back to them later and make any necessary adjustments.

In terms of the processes we used to achieve our plans, we had a fairly smooth experience except for when it came to generating ideas for our game. Figuring out a concept that would effectively teach players about Git proved to be very challenging, and it took much longer than we had anticipated. Despite this, we were able to come up with a solid plan and stick to it, which helped us to create a testable prototype in the end. When it came to the development phase of the project, we faced some challenges but were able to overcome them through effective planning and communication. One of the processes we found useful was utilizing merge requests and issues to control each other's code and organize tasks that needed to be completed. We also frequently used pair programming to work together on more complex tasks. Before beginning the development process, we discussed whether it would be beneficial to use a specific development methodology, such as scrum. While these methodologies can help with the efficiency and quality of the development process in teams, we ultimately decided to not follow any specific methodology. As we are only two people on the team and live in the same dorm, we were able to easily work together and update each other on our progress. Because we have worked on many projects together in the past, we were confident in our ability to work effectively without necessarily following a formal methodology. This proved especially important given the limited time we had for development and the need to be efficient.

In this project, we gained a deeper understanding of the importance and complexity of the planning, design, and development process. As developers, we tend to focus on the technical aspects of building a product, but we learned that the design process can be just as challenging and requires a great deal of time and effort. The task given to us proved to be larger and more complex than we had anticipated, and we could have benefited from having more time to work on it. In particular, the design process was more extensive and challenging than we had expected. Despite this, we did not have the luxury of additional time, so we had to rely on the processes we had established and trust that they would help us to complete the project successfully. Overall, the project was a valuable learning experience that taught us the importance of thorough planning and effective communication.

4.4.3 Architecture

In the prestudy (see Section 2) we found that architecture is important to manage the complexity of game development, and it should be modifiable to support new gameplay mechanics. To allow for this modifiability we decided to base the game architecture on the Model-View-Controller (MVC) pattern. According to Bass et al. (2013), the MVC pattern separates the application into three components:

- **Model** - containing the game data
- **View** - containing the user interface, which displays the data and allows for interaction by the player
- **Controller** - containing the logic for the application by connecting the *view* with the *model*

In essence, the MVC pattern allows the game logic to be separate from the data and the user interface, which gives freedom to modify the user interface separately from the game logic. The MVC pattern is used in our game as seen in Figure 26.

`GameData` is the model in the game consisting of data for managing stats, game states and other data object that are needed in the game. The `GameController` is the main controller which is

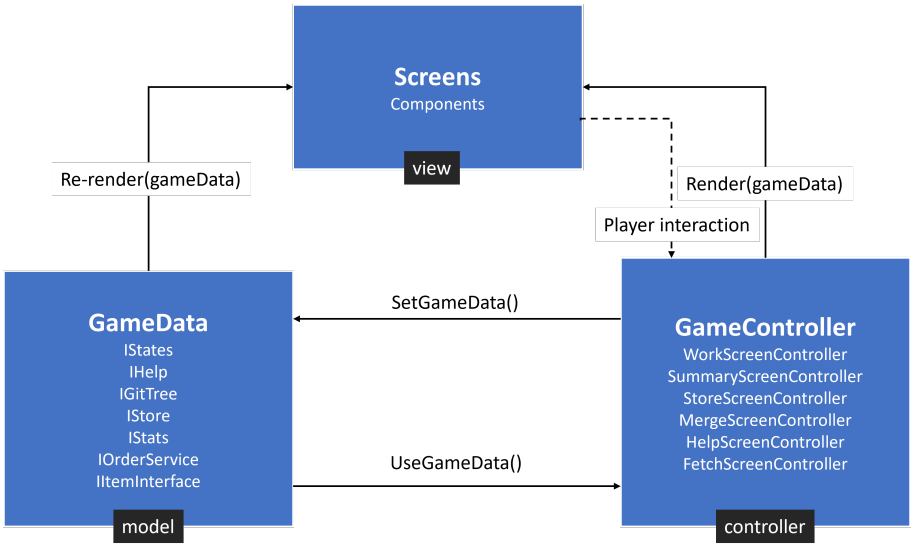


Figure 26: The MVC pattern used in the game, consisting of GameData (model), GameController (controller) and Screens (view).

in charge of delegating control to different controllers depending on the game state. For each game state the `GameController` delegates to different screen controllers, which are responsible for rendering the correct game screen. Each screen controller can also consist of smaller controllers for managing logic within the screen. All the controllers contain functions for manipulating the `GameData`, using `SetGameData()`, which are called whenever the player interacts with components in the view. The view consist of multiple screens, which contain different components, with only one screen rendered at a time. Whenever the `GameData` updates, the view will re-render with the new model.

The MVC pattern makes the user interface of the game easily modifiable, as the game logic is not dependent on the user interface. It is also easy to add new functionality to the game, as it usually only requires creating a new controller which is responsible for the functionality and a new screen or component which are responsible for user interaction with the new functionality.

5 Further work

In order to improve the game and make it more useful for players, there are several areas that need attention in further work. One important priority is to improve the tutorial, so that it is more comprehensive and user-friendly. This could involve revising the existing tutorial content, as well as adding new tutorial sections to cover topics that are currently missing. Additionally, the game's story needs to be written to provide a narrative for the player. The system to implement the story is already present in the game, but it currently lacks content. Lastly, the game should be thoroughly tested and any bugs that are discovered should be fixed.

Another area for further work is to add new functionality for merging and collaborating with co-workers/bots. This is important because working with others is an important concept in Git. This feature could include allowing players to assign co-workers to work on orders in a different branch, as well as tools for resolving conflicts and merging changes. Furthermore, it would be valuable to provide better support for various Git commands, so that players can use the game as a way to learn and practice these commands.

Finally, it would be interesting to examine the potential for using the game in education as a way to promote learning experiences. This could involve conducting research to assess the effectiveness of the game as a teaching tool, as well as exploring potential improvements to make the game more educational or enjoyable. Overall, these are just a few examples of the areas where future work could focus in order to make the game even more valuable and engaging for players.

References

Annetta & Leonard (2010), 'The "i's" have it: A framework for serious educational game design', *Review of General Psychology* **14**, 105–112.

Avila Pesantez, D. & Rivera, L. (2017), 'Approaches for serious game design: A systematic literature review', *Computers in Education Journal* **8**.

Bartle, R. (1996), 'Hearts, clubs, diamonds, spades: Players who suit muds'.

Bass, L., Clements, P. & Kazman, R. (2013), *Software Architecture in Practice, Third Edition*. ISBN-13: 978-0-321-81573-6.

Callele, D., Neufeld, E. & Schneider, K. (2005), 'Requirements engineering and the creative process in the video game industry'.

Chacon, S. & Straub, B. (2022), *Pro Git*. Available at: https://git-scm.com/book/en/v2.

Chorianopoulos, K. & Giannakos, M. N. (2014), 'Design principles for serious video games in mathematics education: From theory to practice', *International Journal of Serious Games* **1**, 51–59.

Dondlinger, M. J. (2007), 'Educational video game design: A review of the literature', *Journal of Applied Educational Technology* **4**.

Ibrahim, R. & Jaafar, A. (2009), 'Educational games (eg) design framework: Combination of game design, pedagogy and content modeling'.

Laine, T. H. & Lindberg, R. S. N. (2020), 'Designing engaging games for education: A systematic literature review on game motivators and design principles', *IEEE TRANSACTIONS ON LEARNING TECHNOLOGIES* **13**.

Lameras, P., Arnab, S., Dunwell, I., Stewart, C., Clarke, S. & Petridis, P. (2016), 'Essential features of serious games design in higher education: Linking learning attributes to game mechanics'.

Lazzaro, N. (2008), *Four Fun Keys*. In Isbister, K., and Schaffer, N. (2008). Game Usability (1 ed.). Chapter 22.

Malone, T. W. (1980), 'What makes things fun to learn? heuristics for designing instructional computer games'.

Qian, M. & Clark, K. R. (2016), 'Game-based learning and 21st century skills: A review of recent research', *Computers in Human Behavior* **63**, 50–58.
URL: *https://www.sciencedirect.com/science/article/pii/S0747563216303491*

Schell, J. (2008), 'The art of game design: A book of lenses'.

Shute & Valerie (2011), 'Stealth assessment in computer-based games to support learning', *Computer Games and Instruction* **55**.

Sweetser, P. & Wyeth, P. (2005), 'Gameflow: A model for evaluating player enjoyment in games', *ACM Computers in Entertainment* **3**.

Tahir, R. & Wang, A. I. (2019*a*), 'Codifying game-based learning: Development and application of league framework for learning games', *The Electronic Journal of e-Learning* **18**, 69–87. available online at www.ejel.org.

Tahir, R. & Wang, A. I. (2019*b*), 'Insights into design of educational games: Comparative analysis of design models'.

Wang, A. I. & Nordmark, N. (2015), 'Software architectures and the creative processes in game development'.

A LEAGUÊ

Learning	Environment	Affective-Cognitive Reactions	Game Factors	Usability	UsÊr
L1. What are the learning objectives of the game?	**E1**. What technical aspects are required for the game to work and best support learning?	**A1**. How can the game provide enjoyment to the users?	**G1.** What are the game objectives to integrate learning?	**U1.** How interface of the game is made easy to use for target users?	**Ê1.** What are the attributes of the target users of the game?
L2. Which learning strategies are being used to enable learning through the game?	**E2**. What is the context for playing the game for learning?	**A2**. How can the game engage the users?	**G2**. What narrative is used to make game compelling and integrate learning?	**U2**. How does the game provide easy learnability to its target users?	**Ê2.** Which cognitive needs (of target users) are considered in the game?
L3. What learning content is being used in the game for target users?		**A3**. How can the game motivate the users?	**G3.** What mechanics are used to make game compelling and support learning?	**U3.** How does the game provide satisfaction to its target users?	**Ê3.** Which Psychological needs (of target users) are considered in the game?
L4. What learning outcome(s) can be acquired from the game?		**A4.** How can the game generate flow?	**G4.** What resources are provided to the users to function effectively that also support learning?		
			G5. What aesthetics are used to make game compelling for target users?		
			G6. What game play is used to make game compelling for target users and support learning?		
Strength of the analyzed game					
Weakness of the analyzed game					

Figure 27: The primary form of the LEAGUÊ analysis tool, which highlight the strengths and weaknesses of an educational game. Picture from Tahir & Wang (2019*a*).