Project Texting Spree

The following document is meant to help the reader understand the premiss of this project better as well as define the main focus of the project as well as its core systems. There are also systems going to be defined that are nice to have but not crucial for the success of the project as a whole as well as the good handling and feeling of the software.

## **Starting Point**

In this day and age, it is only too common for bad stories to be written. Not only do mainstream movies portrait such garbage but also the game industry is hit by this crisis. The stories written are often boring and there is no engagement for the reader to actually want to know the characters portrayed in these stories. Speaking of characters, the stories I am talking about often have no compelling plot and the development of character stagnates to the point where we get only one-dimensional characters, that do not resemble a normal human in the slightest.

Another point is, that many people today do simply not enjoy reading through a story and think to themselves “If I was in his/her place, I would have done it like this!”. And this is where this project steps in. The project codenamed “Texting Spree” is a playable story in the format of a novel. It aims to tell an interesting and compelling story and motivate players to read the story as well as invest themselves into the world, characters and lore and make decisions that shape the rest of the story plot in a way the player actually wants.

So, depending on the players choices, other options are opened to the player and the story progresses differently for every decision made. Other than that, there are other systems going to be implemented, such as and inventory where the player can pick up items that will hinder or help him progress the story.

## **People involved**

In the following chapter we are going to have a look at all the people who are going to be involved in making this project a reality. First of all, the following people are going to be named “Three Trolls Games” as a pseudo-company that will form somewhen in the future. The table below lists all the people involved in the project as well as their titles within the future company.

|  |  |  |
| --- | --- | --- |
| **Person** | **Title** | **Description/Work area** |
| Bärfuss Frederic | Lead Story Designer | The lead story designer is responsible for the story plot that will be used in this project. His primary focus is to make sure that the plot has no inconsistencies and plot holes. |
| Dulla Kastriot | Game Director | The game director is the project leader. He has the responsibility for the entire project, its success or downfall. His work area is to make sure that every other participant of the project has what he needs to fulfil their role as good as possible and also, he will help wherever he can.  The game director is also the primary decision maker in the project and other lead roles have to converse with him first before changing any specifications of the project. So, it’s the game director that approves or declines features and he oversees changes to the story. |
| Dulla Kastriot | Programer | Programmers are responsible for the technical implementation of the features approved by the game director and the lead programmer. They will primarily work with the source code and its components. |
| Dulla Kastriot | Assistant Story Designer | The assistant story designer assists the lead story designer to write the story of the project. He makes sure that there are no inconsistencies or plot holes in the story. |
| Dulla Kastriot | Narrative Designer | The narrative designer is responsible for writing the overall plot of the game as well as write the characters that are present in it. It is his doing in how characters are written and presented within the overall world of the project. |
| von Ballmoos Kevin | Lead Programmer | The lead programmer is the chief of programming and responsible for the technical implementation of the project. He is primarily in touch with the game director and works closely with his team to make sure the code is reusable and stable. |
| von Ballmoos Kevin | Lead Game Designer | The lead game designers’ job is to make sure that the game elements are coherent and make sense. He is responsible for the usability and user friendliness as well as the overall design of the project. |

Table 1: Job descriptions

During the project, there can be other roles that need to be filled. These roles will therefore be added to the table above and the document will be updated.

## **Development and Engine**

The development of this project is primarily made with the Unity Engine (version 2021.3.5f1 or higher). The components that ware to be developed or used in this project are going to be downloaded from the Unity Asset Store and are to be used as placeholders for the rest of the development process. Once the core systems and functionality has been established, own assets are going to be created and used instead of the aforementioned Unity Assets.

The code as well as the developed C# scripts are going to be created with the IDE known as Rider (version 2022.2.2 or higher). If the individual programmer desires, he can change the IDE to one of his likings. If the Lead Programmer decides not to use any other IDEs for compatibility reasons or otherwise, Rider is to be used in all cases.

## **Core features**

In the following table the core systems are defined as must-have functions and are therefore to be implemented into the final game under all circumstances. In the following table we can see a name as well as a description for all the functions made. The more important part here, however, is the status of development that can be seen.

|  |  |  |
| --- | --- | --- |
| **Naming** | **Description** | **Status** |
| Dialogue System | In this system we can create multiple dialogues that are used in the game to progress the story. This is going to be the one core system the project depends most on.  The player is going to be able to make choices that will progress the story in a different direction, based on the decisions made by the player, the dialogue takes another turn and therefore can be either a game over for the player or it can lead to a new chapter, treasure or item.  The player should also be able to go back in the dialogue to be able to read the story that up until the last checkpoint. | Done. |
| Save System | The player should be able to save the game at any point in the story. All his decisions as well as his items and the page of the story where he is at are to be saved securely.  Eventually we can build in an auto save function that saves the game at certain checkpoints. | Done. |
| Loading System | The player is able to load one of the made saves including all of his choices as well as his inventory and current status. | Done. |
| Title Screen / Menu Screen | There must be a title screen developed for the game. In this screen all the normal menu points should be available, for example “New Game”, “Continue”, “Load Game” as well as “Options” and “Quit Game”. | Done. |
| Inventory System | An inventory should be implemented that allows the player to pick up items during the playthrough. These items can manipulate the choices made and unlock or lock certain choices for the player. These items should be visible in a separate menu and a tooltip with a description is going to be needed for the individual item as well as a number indicating how many of said items are in the inventory. | Done |
| Checkpoint System | During the game there should be multiple checkpoints where the game is automatically save. These checkpoints are going to be spread throughout a chapter so that the player can easily reload the game in case of death or when he is stuck in a situation that he does not want.  A checkpoint could probably be set when a very important decision is going to be made by the player. | Done |
| Pause menu / Options menu | By pressing the “Esc”-Key on the keyboard the player should be able to access the pause menu. This menu is basically a copy of the already implemented menu with the addition that the player can chose to go back to the title screen instead of quitting the game completely. | In development. |
| Maps and Drawings | Throughout the game there should be multiple maps for the player to be able to see where he is at the moment. Especially at the beginning there should be a world map to show the players in what kind of environment he is.  In later sections of the game there should be maps of cities, villages or even drawings of different rooms or characters. A good example here is found in the game “Pillars of Eternity: Deadfire” where the book shows a clear picture of the environment on one side, while telling the story in the other side. | In development. |
| Main character | At the beginning of the game the player must choose between three main characters who all have a different prologue but follow the same story in a different way. Depending on the choice different dialogues are triggered. | In development. |
| Music | In the prototype build of the game, there are going to be at least 3 soundtracks that depict an idle state as well as a more sinister and more light-hearted state of the story. Once the project has advanced to a more advanced state there can be an idea for more soundtracks as well as sound effects that are played during the game for example depicting a large-scale battle or a lively tavern. | In development. |
| Game Screen (GUI) | The game screen. Its just that. The actual game. | In development. |

Table 2: Core features

## **Advanced features**

In the following chapter, we define the systems and functions whose development contributes significantly to increasing the quality of the project, but is not functionally mandatory. These functions can, but do not have to be built in. They are therefore defined as nice-to-have features.

|  |  |  |
| --- | --- | --- |
| **Naming** | **Description** | **Status** |
| Combat System | A combat system should be built in, in which the player is able to lead a fight against different opponents. What the combat system should look like has not yet been decided. | Not planned for development. |
| Interactive Map | A map that allows the player to move around it at will. But it also provides him with important information such as the nearest settlement or points of interest. | Not planned for development. |
| Trading System | The player can contact various merchants to sell found items and buy other items. A currency is therefore also integrated into the game. | Not planned for development. |
| Abilities | Depending on the protagonist, the player has a number of abilities that allow him to make it through situations or decisions more easily or even allow him to make decisions in the “best possible” way. | Not planned for development. |
| Character Creation System | The player is able to create his own character including his own name and gender. | Not planned for development. |
| Character Attributes | The player has various attributes such as strength, dexterity, endurance and charisma, which allow him to make various decisions more easily. This system would allow a very dexterous player to easily sneak past guards, while a strength-based character would fight them head on. The magic-based character would simply become invisible or blind the guards with an illusion. | Not planned for development. |
| Day-Night-Cycle | A system that records the day and night time of the world. Based on time-of-day certain character might not be seen or | Not planned for development. |
| Event Director | An invisible entity that, based on various decisions or systems such as the time of the day or the interactive map, triggers events that affect the course of the game. | Not planned for development. |
| Audio | Audio for the game. This includes music and various audio effects. | Not planned for development. |
| RNG-System | Various decisions can have a negative impact on the ongoing story, when failed. For example, there is a small chance of falling off a roof during a scene when you try jumping from it. | Not planned for development. |
| Hunger and Thirst System | The player gets hungry or thirsty during the game, depending on how long he has gone without these things, he might start to experience negative effects. Sleep could also be a part of this system. The time system is essential for this. | Not planned for development. |
| Language support | The game should be in German/French/Albanian and Chinese. | Not planned for development. |
| NPC Party Members | The player can find several party members during the game and invite them to his group. These can help him in battle or give quests or useful information or even alternative routes for existing quests. | Not planned for development. |
| NPC Party Members Portraits | NPC party members or important characters have their own portrait. | Not planned for development. |
| Camp/Shelter? | For interacting with the party's NPC characters there is a camp or a shelter where the party can retreat. | Not planned for development. |

Table 3: Nice-to-have features

## **Goals**

The aim of this document is to provide a solid basis for the documentation of the project and to provide members with an insight into the future and the current status of development.

Furthermore, it is the goal of the developers or "Three Trolls Games" to complete an executable prototype of the project including all must-have features by December 31, 2023. Animations, assets, design and audio are not necessary.

## **Code and versioning**

Along with the assets, the code is the heart of the project. The code is basically only edited by the programmers or the lead programmer. Various standards are to be applied in relation to the syntax, the comments and the creation of methods and classes. These standards are listed in the separate document “Coding Conventions”. The conventions written therein must be used under all circumstances and may only deviate in cases approved by the game director or the lead programmer.

The code is always versioned using GitHub. In addition to the master branch, there is also the develop branch within the version control. For all further developments on the project, so-called feature branches must be created. The following rules apply when creating feature branches:

1. Each new feature branch is written in lower camel case and must have the word «feature» first. **Example**: feature/mainMenu.
2. The feature branches are separated from the actual code structure using a backslash. That means it is named **feature/<work>**. **Example**: feature/mainMenu.
3. The bug branches are separated from the actual code structure using a backslash. That means it is named **bug/<BugName>**. **Example**: bug/storyNodeNotWorking.
4. All feature branches are created from the development branch. The reason for this is that most of the features are on the development branch and the development is the most advanced.
5. After the work is done, a pull request must be made to the lead programmer or to the game director. A new branch with a new feature can only be created after confirmation by one of the two.
6. After the pull request has been made, the programmer's work is done and he moves on to a new task. No merges may take place on the development branch. Only the lead programmer or the game director have the authority to perform a merge on the development or master branch.
7. As soon as the branches have been merged onto the development or master branch, the feature branch is deleted.

## **GUI development**

GUI development is a high risk in this project for multiple reasons. One of them being, that the GUI is a delicate matter and that Unity handles a lot of metadata for each piece of the GUI. To minimize the risk of bugs and failure of the entire program we have implemented the following rules in terms of the development if GUI elements.

In the versioning there is a branch called **feature/GUI** this is the only branch on wich GUI changes are made and saved. Every other branch should not contain any GUI changes for the risk of clashing merges when merged into other feature branches or even the development branch.

If one must make changes on the GUI a new branch is create with the name **feature/GUI<work>. Example**: feature/GUIpauseMenu.

The document “Coding Conventions” can be consulted for a more detailed documentation of the code standards as well as the naming conventions during the project.

Twice a week, every developer has to merge his branch into the “feature/test” branch. Here all features are going to be tested as well as the new GUI changes will be reviewed. Afterwards this branch is going to be merged into the development branch.

**Disclaimer**: All code written in this project and subsequent documents, belong to the project founders and members of the board.

* 1. **XML story files**

The story is to be made up of several chapters. Writing the chapters yourself in the boxes of the dialog editor is incredibly tedious and can quickly lead to confusion. This inevitably leads to making mistakes within the story and thus not being able to ensure the desired quality.

To counteract this problem, we have devised an XML structure that is able to be read by the dialog editor and to save the individual story chunks and chapters in the respective nodes. In order to enable this in the code, however, the file must have a special format and a special name so that the code is able to fill in the story correctly. Below is an example of the XML structure that must be created so that the file can be read correctly. A more detailed description as well as another example can be found in the document “Coding Conventions”.



Figure 1: XML structure

## **Logging**

Logging is a big part of debugging the code. As for now, we have no method to actually debug the code efficiently. Using **Debug.Log()** all the time is first of all time consuming and secondly extremely dumb to do, because during testing, these logs can cause issues or be forgotten to actually take up calculation time for the engine.

Therefore, we have implemented a Logger class that logs the state of the game. For more detailed information on how to use the class, consult the document “Class Documentation” in the project documentation folder.

## **Sprite and labelling issues in GitHub**

To ensure the best possible production of the game, we have created a sprite sheet in GitHub. Using this, we can create tasks as well as issues to divide the work between the programmers and start working on the individual tasks one by one.

The following example shows how to actually get to the project and how to get to the sprite. First of all, you need to have an internet connection as well as an active GitHub account for either the lead programmer or the game director to invite you to the project in the first place. Once on GitHub, follow the steps provided in the screenshots.

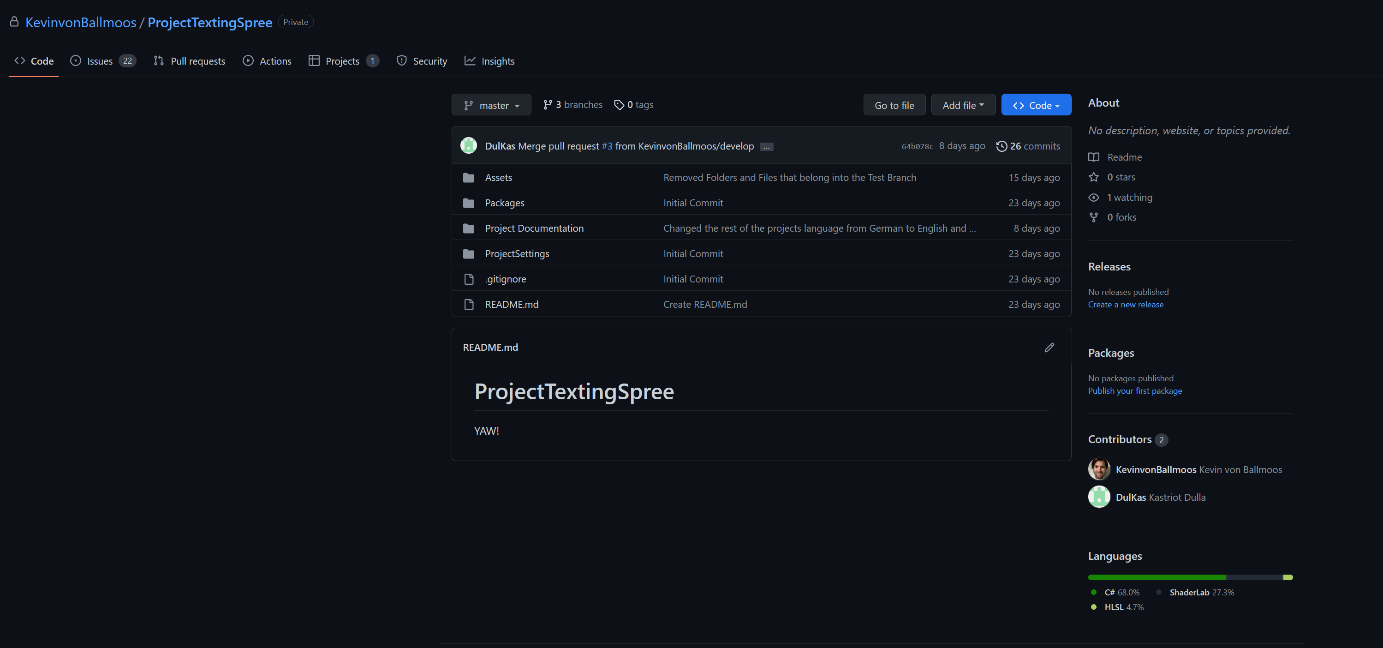


Figure 2: Go to the project page

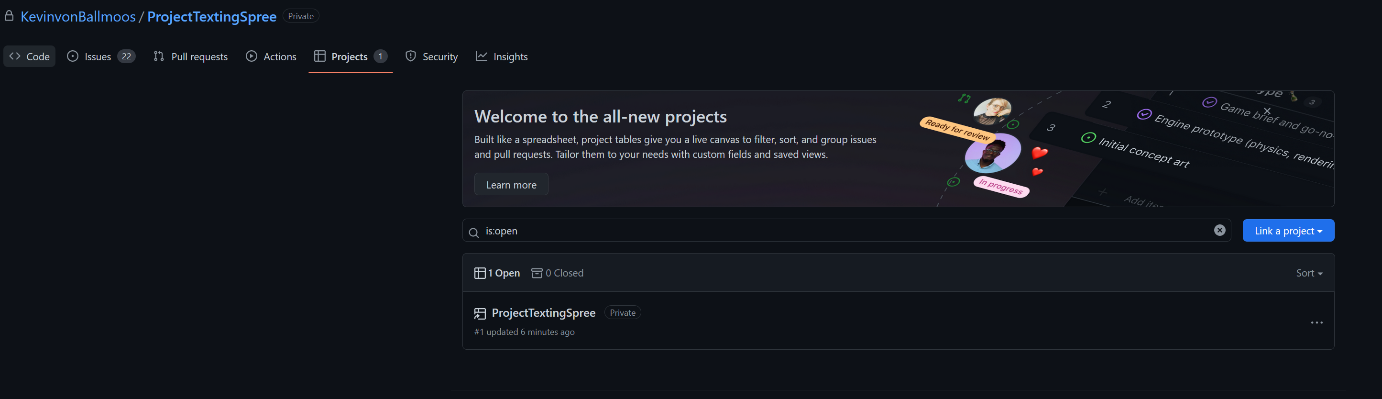


Figure 3: Navigate to the "Projects" tab and click on the project below.

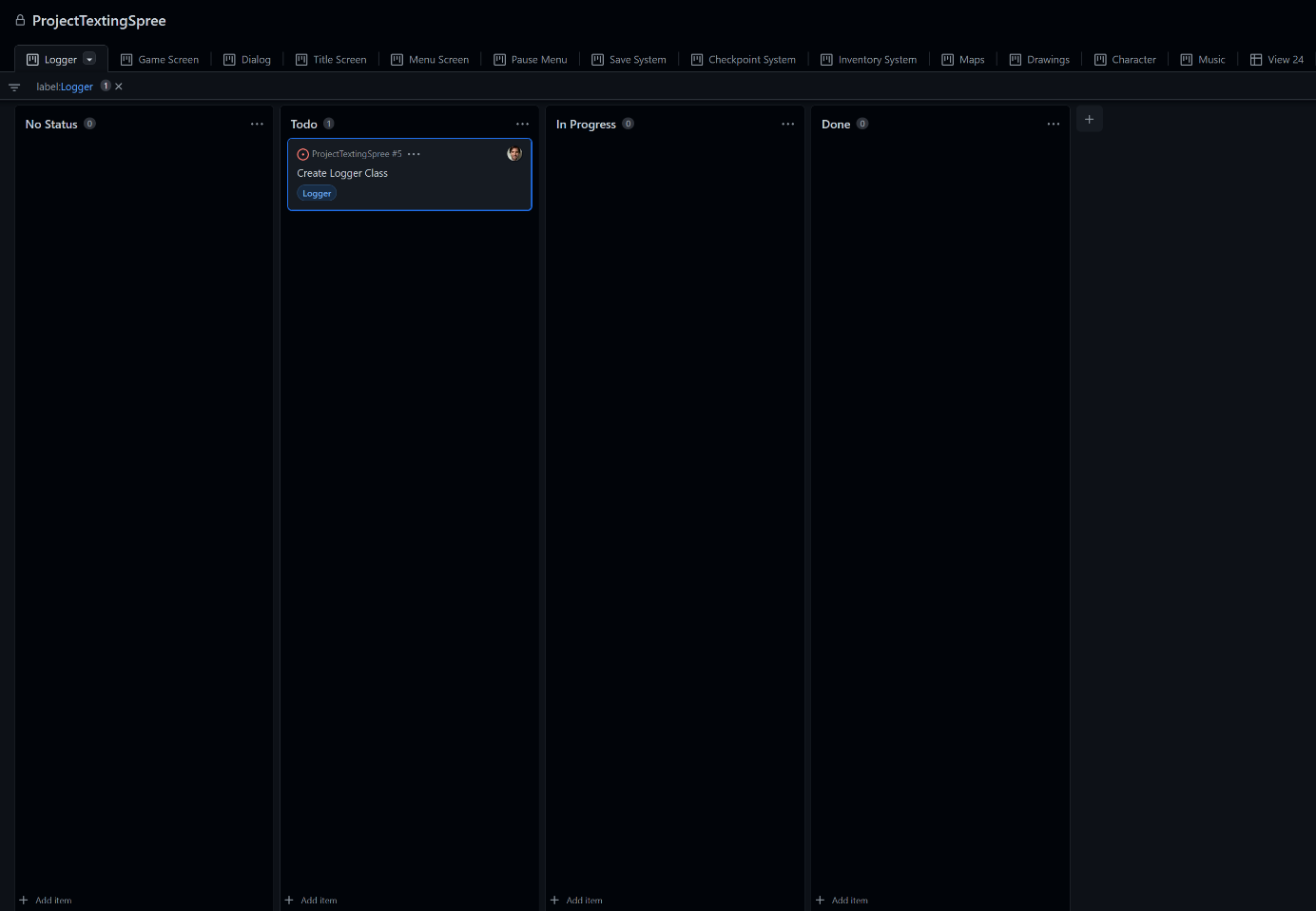


Figure 4: Once on the project clicked, you should be in the sprite and see the different tabs of the core features

This is the sprite as well as all the tabs that represent all the core features that are going to be created during the project. All core features and later advanced features are going to be created as issues in this project page.

If one needs to create a new task for himself or another programmer then the following steps must be maintain in order to be able to make the workflow coherent and not lose focus on who is working on what.

1. First of all, navigate to the view simply known as “Backlog”.
2. In the Sprite “No Status” click the small button at the end of the page saying “Add item” and create a draft item.
3. Enter the name of the draft.it should appear at the end of the sprite and all other issues here.
4. Click on the draft and give it a description with the tasks that you need to perform. Be very specific about what you need to do and describe the steps that you want to take to get the task done. Herby use the - [ ] provided by GitHub to create checkboxes that you can check once the step has been completed. This way the lead programmer can monitor your work and can help on a specific task once you are stuck or away.
5. Once the description and the steps have been defined, convert the draft to an issue using the “Convert to issue” button on the tasks description.
6. Add it to the “ProjectTextingSpree” project.
7. As soon as you are done, click the “Add labels…” button on the right of the screen and add the label of the feature as well as the “task” label.
8. Once this is done, you can switch to the view of the respective feature and you should see your task in the “No status” sprite. Immediately drag it to the “Todo” sprite or the “In Progress” sprite, if you plan on working on it immediately.

Please bear in mind, that only the lead programmer and the game director respectively can create feature issues. It is usually a good idea to consult with the lead programmer before creating a task on your own.

## **Mock-up**

In the following section we see the mock-up idea of how the game will look in its prototype state. This mock-up is by no means final and is subject to change. Should there be new mock-ups drawn, the old ones have to be marked by a clear sign that they are old versions and therefor no longer in use. No mock-up is to be deleted from this document.

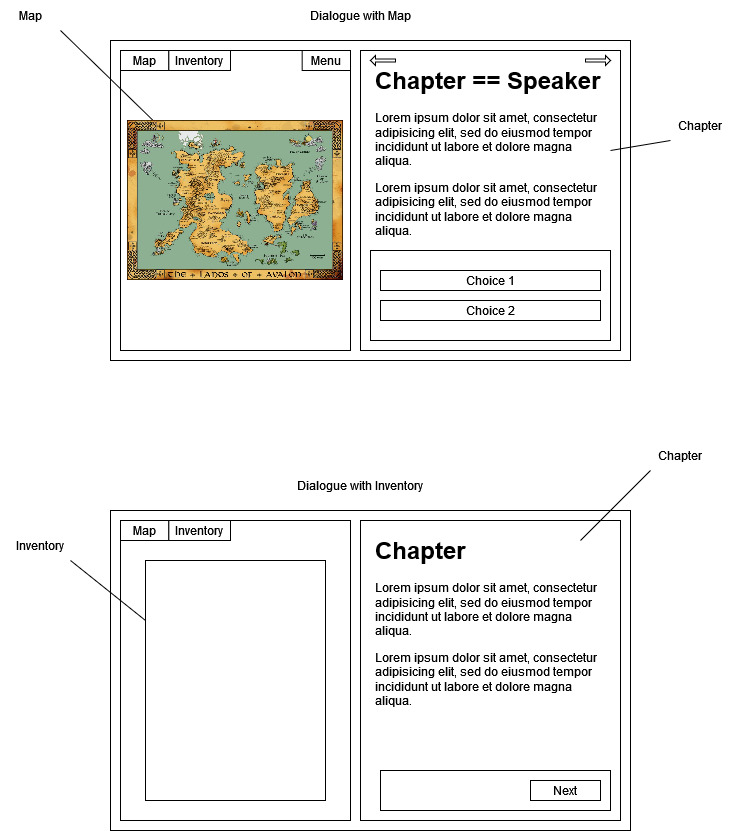


Figure 5: Mock-up main screen 0.0.1

## **Finances and Compensation**

The project, code named “Project Texting Spree”, is a pure passion project and therefore offers **no** monetary compensation to all of the members of this project.

For financing the needed assets as well as pictures or audio during the development of the project, all costs are split between the members equally. It is seen as mandatory to split the bill in order for the project to be somewhat successful and any member who does not agree with this policy is excluded from the project. Eventually, there is going to be an attempt on setting up a Kickstarter campaign or a GoFundMe campaign in order to ease the financial burden on the members of the project as much as possible.