Forward Software Design

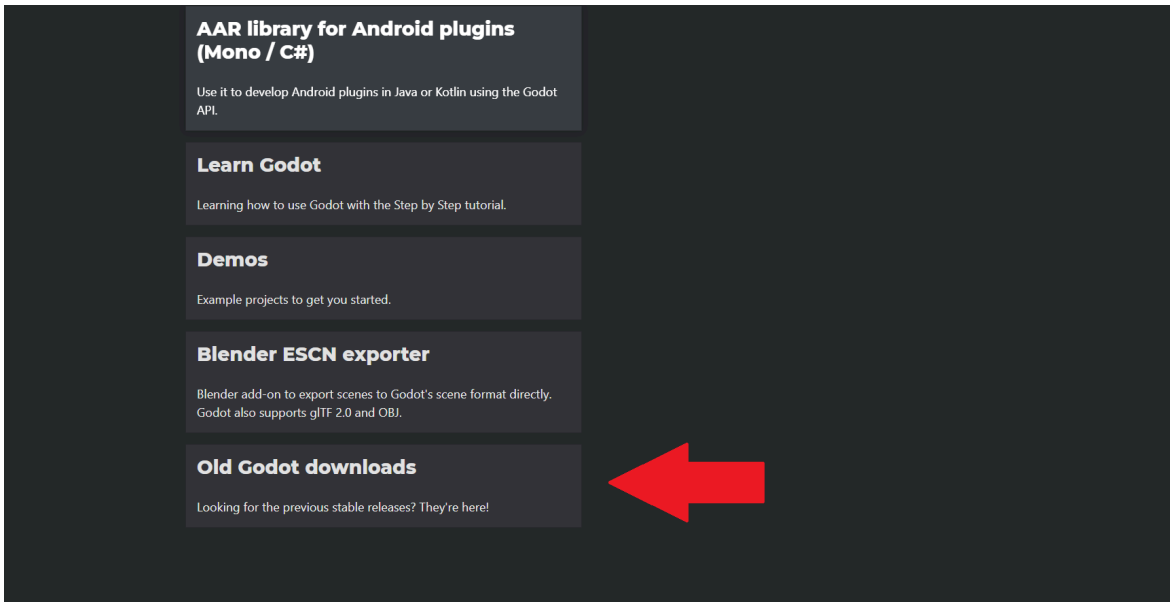
Project Github Repository: https://github.com/Nivrad00/social-landmines

Setting up Godot environment

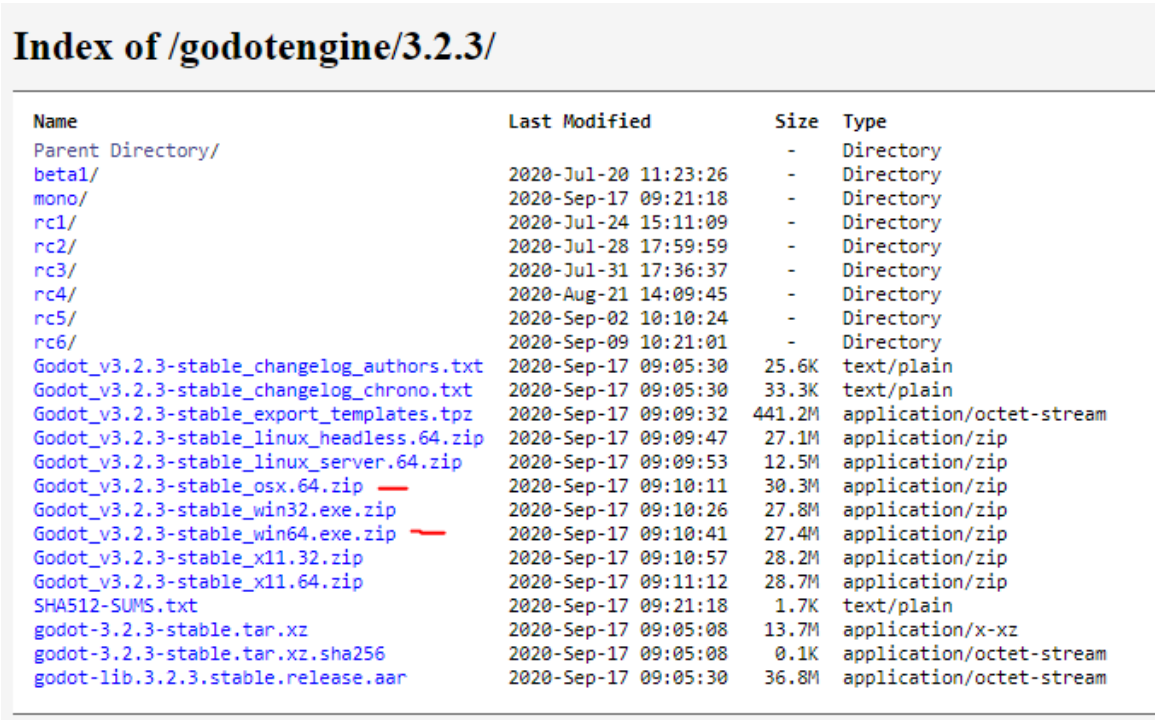
Forward was developed on Godot version 3.2.3. Newer versions have not been tested, but aren't likely to be compatible.

Step 1: Downloading Godot

Access Godot's download page here: https://godotengine.org/download
Scroll down and click on old godot downloads.



A list of previous godot versions should pop up. Click on 3.2.3. Depending on your operating system, download the appropriate zip file.



Index of /godotengine/3.2.3/

Name	Last Modified	Size	Type
Parent Directory/		-	Directory
beta1/	2020-Jul-20 11:23:26	-	Directory
mono/	2020-Sep-17 09:21:18	-	Directory
rc1/	2020-Jul-24 15:11:09	-	Directory
rc2/	2020-Jul-28 17:59:59	-	Directory
rc3/	2020-Jul-31 17:36:37	-	Directory
rc4/	2020-Aug-21 14:09:45	-	Directory
rc5/	2020-Sep-02 10:10:24	-	Directory
rc6/	2020-Sep-09 10:21:01	-	Directory
Godot_v3.2.3-stable_changelog_authors.txt	2020-Sep-17 09:05:30	25.6K	text/plain
Godot_v3.2.3-stable_changelog_chrono.txt	2020-Sep-17 09:05:30	33.3K	text/plain
Godot_v3.2.3-stable_export_templates.tpz	2020-Sep-17 09:09:32	441.2M	application/octet-stream
Godot_v3.2.3-stable_linux_headless.64.zip	2020-Sep-17 09:09:47	27.1M	application/zip
Godot_v3.2.3-stable_linux_server.64.zip	2020-Sep-17 09:09:53	12.5M	application/zip
Godot_v3.2.3-stable_osx.64.zip	2020-Sep-17 09:10:11	30.3M	application/zip
Godot_v3.2.3-stable_win32.exe.zip	2020-Sep-17 09:10:26	27.8M	application/zip
Godot_v3.2.3-stable_win64.exe.zip	2020-Sep-17 09:10:41	27.4M	application/zip
Godot_v3.2.3-stable_x11.32.zip	2020-Sep-17 09:10:57	28.2M	application/zip
Godot_v3.2.3-stable_x11.64.zip	2020-Sep-17 09:11:12	28.7M	application/zip
SHA512-SUMS.txt	2020-Sep-17 09:21:18	1.7K	text/plain
godot-3.2.3-stable.tar.xz	2020-Sep-17 09:05:08	13.7M	application/x-xz
godot-3.2.3-stable.tar.xz.sha256	2020-Sep-17 09:05:08	0.1K	application/octet-stream
godot-lib.3.2.3.stable.release.aar	2020-Sep-17 09:05:30	36.8M	application/octet-stream

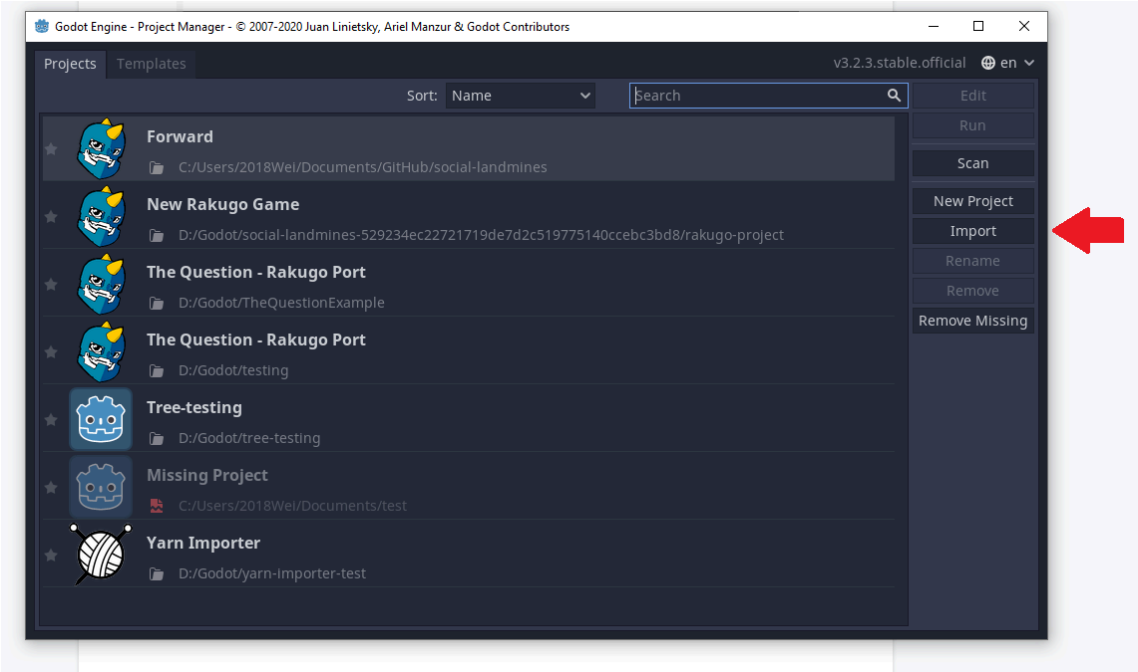
Here are some direct links to the downloads, though they may break in the future:
Windows: Godot_v3.2.3-stable_win64.exe.zip
MacOS: Godot_v3.2.3-stable_osx.64.zip

Unzip the file to wherever you want.

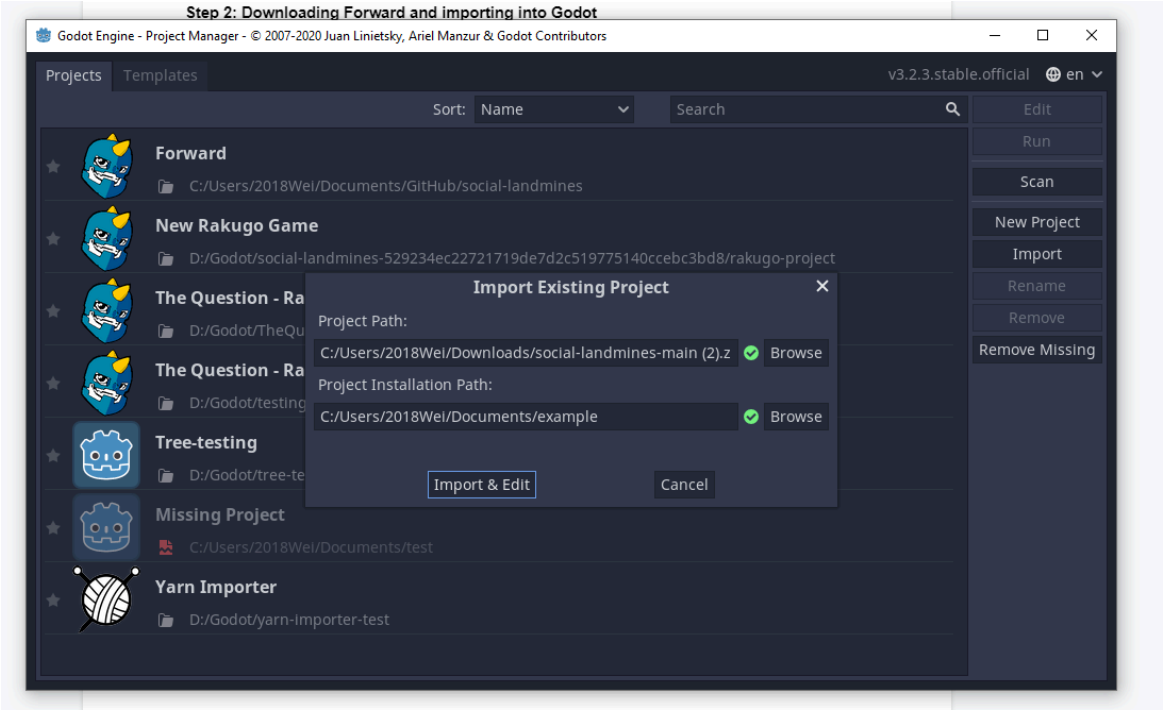
Step 2: Downloading Forward and importing into Godot
Access the github here: https://github.com/Nivrad00/social-landmines

Feel free to clone the repository or download the zip, you just need to know where your repository is / where you downloaded for the next few steps. This walkthrough will go over the process if you downloaded the Github repository as a zip file.

After downloading the zip file from Github, open Godot. On the right hand side should be a button named Import.



Click on the import button and a window should pop up asking you to find the project path. Click Browse and find the location of the zip file you downloaded. After selecting the zip file, it will ask you to specify an empty folder for the project installation path. Create an empty folder anywhere and select it. If you did this properly, the window should look like this.



Click Import & Edit, and the editor should automatically open with the Forward project loaded. It may take a moment to do this, so be patient and it should work fine.

From now on, after you open your Godot executable, Forward will be under Projects and you can proceed to open it from there.

The repository will not be maintained and will need to be forked for future development.

Design decisions

Choosing a game engine

Our first objective was to choose an engine to create our visual novel. There exist engines that are specifically designed for creating visual novels such as Ren'py and Visual Novel Maker, but these engines were limited to the basic gameplay of selecting dialogue options. We wanted to incorporate mini-games within the visual novel, such as coloring or visual aids to help with breathing, which were either outside the scope of these visual novel engines or inconvenient to implement.

Eventually we agreed to using Godot. We decided on using Godot due to two of our group members, Darvin and Wayne, having previous experience working with Godot. The scripting language of Godot, GDScript, is also closely based on Python which is relatively easy to pick up. The documentation for GDScript is community developed and can be found here: https://docs.godotengine.org/en/stable/getting_started/scripting/gdscript/gdscript_basics.html.

Godot allowed the ability to incorporate the mini-games we wanted into our visual novel relatively easily, so we thought it was the best choice for our project. Godot also allows easy export to any device, as well as WebGL builds. We later found out that the version of Godot that we were using, v3.2.3, did not support threads for web builds, which Forward uses extensively. The latest version of Godot, v3.3.0, has added support for threads for web builds, but we had not had the chance to update our version of Godot to the latest one.

Developers must be familiar with tree/node structure of Godot, and have knowledge of GDScript. Because of how familiar GDScript is to Python, knowledge of Python will also be sufficient. We have developed the game using Godot v3.2.3

Choosing frameworks and using Yarn

Our first decision after choosing Godot as our engine was to make the use of Yarn. Yarn is a language and tool for writing game dialogue. The Yarn syntax found here https://yarnspinner.dev/docs/syntax/ is mostly adhered to. We have also added some custom syntax that can be found here: https://docs.google.com/document/d/12xAfthQDePbqiCl3WboHg5VmQoYLBEpeqsxNl4aJy8s/edit?usp=sharing

We wanted to use Yarn because how it allows for the branching of narrative and dialogue options, which is essential for visual novels. Most importantly, Yarn has its own Yarn Editor that has an easy-to-use interface. All the dialogue for the game was written by Dr. Marraccini's research team, so using Yarn allowed them to write their branching narratives with minimal coding experience. We found an addon for Godot called Yarn Importer that allowed us to interface with Yarn files (extension .yarn) in Godot.

Working with Rakugo and Yarn Importer

Rakguo is an open source visual novel framework built as an add-on for Godot, which can be found here: https://github.com/rakugoteam/Rakugo. It contains most of the functionality that a visual novel would have such as a main menu, choice buttons, dialogue boxes, and saving/loading. We decided to use Rakugo so that we did not have to make these features from scratch, saving us a lot of time moving forward in our development. Rakugo has its own custom-made functions that are documented here: https://rakugodocs.readthedocs.io/en/latest/.

We have made a variety of modifications to Rakugo to fix bugs, such as correctly remembering state when using the "rollback feature" and correctly resetting state when exiting to the main menu. We also extended Rakugo's GUI to implement the mood thermometer, "coping strategies" dropdown menu, and minigames.

Yarn Importer is an open-source tool built in Godot to parse Yarn files and run through the script, outputting speech and choices to a graphic interface. The Github for Yarn Importer can be found here: https://github.com/naturally-intelligent/godot-yarn-importer.

Yarn Importer can handle branching to different Yarn scenes, dialogue and choices but doesn't have functioning support for logic statements in Yarn and a few other features of the language. We have added this additional support by adding an expression parser, a basic environment, and additional logic to Yarn Importer's parsing. This includes "set" statements which set variables in the environment; a conditional system using if, else, elseif, and endif tags; the ability to unconditionally jump from one node to another; and commands for setting the background and the characters in the scene.

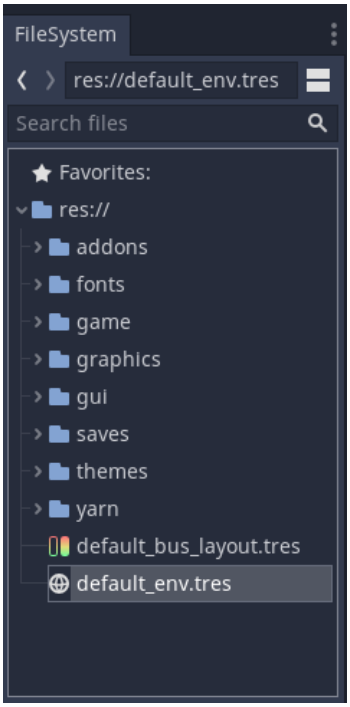
On top of these additions to Yarn Importer, we've created a script called yarn-rakugo-interface.gd that extends yarn-importer.gd to interface with Rakugo.

Choosing not to use dynamic GDScript generation

Early in development we had to consider how to use yarn-importer alongside Rakugo. One of the ideas was to use yarn-importer's "export to GDScript" function and dynamically create dialogue nodes that would populate the main scene tree before the game begins. This pre-processed Yarn script would use the syntax detailed in Rakugo's documentation.

Instead, we decided to proceed with the method our final product is using, which is to iterate through the Yarn script and have it interface with Rakugo's functions in real time. We decided on this since one of our members was working on making yarn-importer work with Rakugo, and their implementation would have been made easier with the method we decided on.

File System and Structure

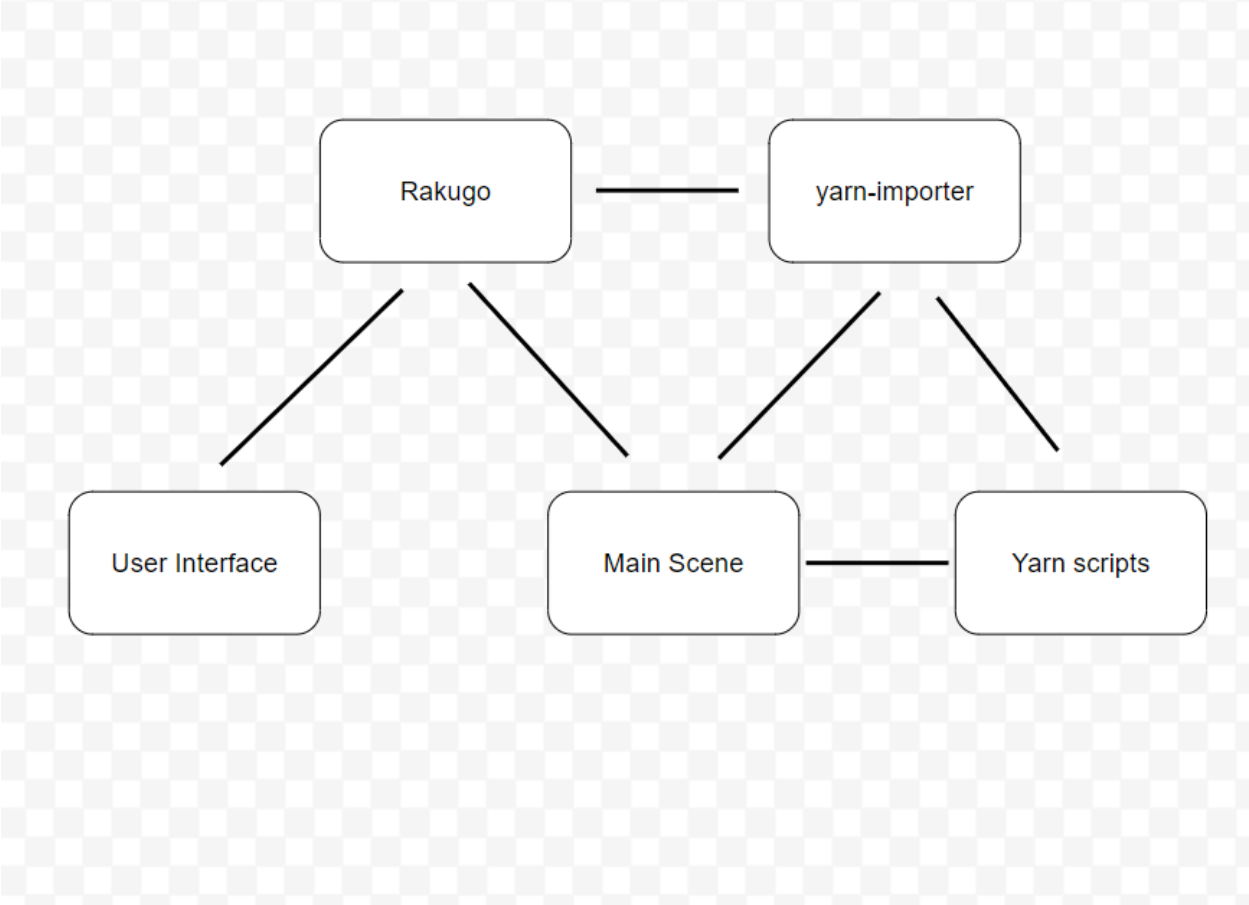


- res://addons contains all code related to the Rakugo framework.
- res://fonts contains fonts that are used by Rakugo and the game.
- res://game has three sub folders:
    - res://game/audio contains all audio files.
    - res://game/characters contains a folder for each character shown in the game. Every character has their own folder with all image assets inside, as well as the Godot scene pertaining to the character. There is also a folder for backgrounds, which are treated the same as characters in Rakugo. All new character scenes must be added to the main scene manually and follow the setup here https://rakugodocs.readthedocs.io/en/latest/tutorials/show_and_hide/
    - res://game/main_scene contains the main Rakugo scene, as well as all the scripts used for the questionnaire and end screen, which are children of the main Rakugo scene. Note that in Godot, the term "scene" is used to refer to a collection of hierarchical nodes in a .tscn file, while in Rakugo, a "scene" is a special type of Godot scene that represents one chapter in a visual novel. We

elected not to use Rakugo's scene system, and rather built the entire game within a single Rakugo scene -- namely, main_scene.tscn.

- res://gui contains all user interface scenes and scripts that are used in the game. The minigames, coping strategies menu, and thermometer are also located here, under InGameGui.
- res://saves is a folder that used to contain the save files for the game. This folder should now be defunct -- saves are stored in user://saves so that they can be written to in the exported builds.
- res://themes contain all the themes used by containers and UI elements (those in res://gui)
- res://yarn contains the expression parser, yarn-importer, and yarn-rakugo-interface. This is also where all the Yarn scripts are located. All scripts must used the *.yarn file extension, but can easily be opened and edited using any text editor or the free Yarn editor at https://yarnspinnertool.github.io/YarnEditor/. Note that .yarn files are not recognized as resources by Godot and thus cannot be opened within Godot. Instead, they can be accessed using a file explorer. Scripts must be linked to each other using the syntax described in the custom syntax document, otherwise they will not load. For example, the final line in script "2 hallway" should be <<load 3 class 1>>

Architecture



Forward is built using the Rakugo framework for visual novels. Rakugo came with a built-in user interface that we adapted for our game. We also extended Rakugo to fix various bugs and to add the coping strategies, thermometer, and minigame features. Rakugo requires a main scene to function, as stated in their documentation. Yarn-importer is used in the main scene to read Yarn scripts and interface them with Rakugo methods.

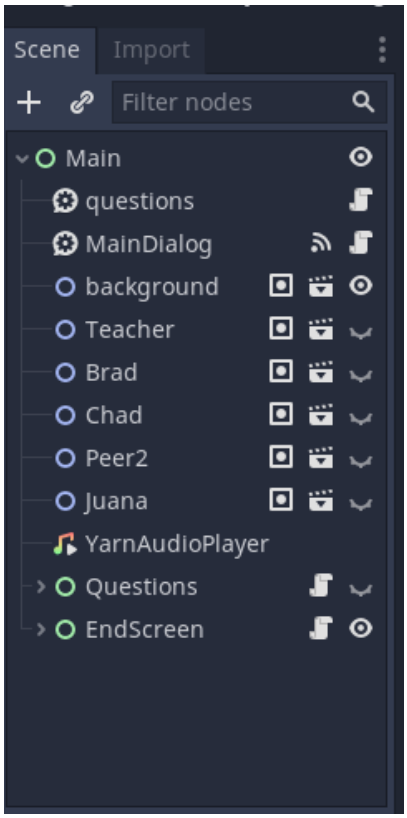
Decomposition

Modules

- Rakugo: The main module that the game is built upon. Most of the game's user interface elements such as the main menu, dialogue boxes, dialogue choices, and buttons came from this framework. We are using Rakugo's save/load, skip, and rollback features. We are not using Rakugo's hide function which hides the user interface while in game. The yarn-importer script translates the Yarn scripts according to Rakugo standards in order to display dialogue and choices. This means using Rakugo's own functions that are in its documentation.

- Yarn-importer: Used to load and read Yarn scripts. We have adapted this to translate into Rakugo's standards. The script for this can be found in res://yarn, referenced in file system and structure. The script yarn-rakugo-interface extends yarn-importer.

- InGameGui: This is a scene included with Rakugo that we modified to contain the coping strategies dropdown menu, the mood thermometer, and the minigames.



- Main Scene: Above is an image of the main scene that controls the scripts that are displayed.
    - "questions" dialogue node: [No longer used]
    - "MainDialogue" dialogue node: decides which Yarn script to start with, then iterates through Yarn scripts using yarn-importer.
    - All Node2D nodes (blue circles): contain Godot scenes that control what image is shown, which Rakugo uses to display images.
    - YarnAudioPlayer: the single audio channel that plays audio for the game as specified by <<start x>> and <<stop>> commands in Yarn scripts.
    - "Questions" control node: the questionnaire that is presented at the beginning of the game.
    - "EndScreen" control node: the screen that is shown at the end of the game

Data

- All variables declared in Yarn scripts, variables created by the questionnaire, and the player's mood are set as global variables that can be accessed at any time. They are set in the script *Global.gd*, which is located in res://game/main_scene. These variables are accessed by yarn-importer, minigames, the thermometer, and the end screen.
- All Yarn scripts are located in the yarn folder, as referenced in file systems and structure. Yarn files should be structured according to yarn spinner syntax, which includes some custom syntax made for Forward.
- Character and background assets that are used in Forward are located under res://game/characters, also described in file systems and structure.
- Save files are made using the Rakugo framework. They are located under user://saves, although Rakugo originally placed them in res://saves. Rakugo makes save files by remembering every dialogue choice the player makes. When Rakugo loads a save file, it performs what it calls a "jump," where it simulates playing through the game from the start of the Rakugo scene, making all the same choices the player made. (Note that the rollback feature also requires making a jump, namely from the start of the Rakugo scene to the line directly before the current one.) We have modified our variable storage to make sure that variables altered by the questionnaire or minigames are preserved as well when a save file is loaded.

Deployment

Note: These are instructions to deploy on Windows. Forward can be deployed to other platforms by using the appropriate export template. However, it cannot be deployed to WebGL, as detailed earlier.

To export, go to Project/Export… to pull up the Export window. Click on Add… to add the export preset corresponding to the build you'd like to create, such as the Windows Desktop build. You will be prompted to download an export template if you haven't already.

Before you export exporting, under resources tab you must add "*.yarn, game/*" to the "Filters to export" field, otherwise the game will crash on execute. The script export mode should be set to "text." This export configuration should be automatically saved the next time you open the export dialog.

To set the application icons correctly on Windows, Godot requires a third-party program called rcedit to be used. See this page for a tutorial.
https://docs.godotengine.org/en/stable/getting_started/workflow/export/changing_application_icon_for_windows.html

We have created Windows Desktop, Mac OSX, and Linux/X11 builds and tested on Windows 10, Mac OS 11.3.1, and Fedora 34.

Modifications

Although we started with the idea that Yarn scripts could be swapped in and out easily, the process of creating, importing, and deploying new Yarn scripts turned out too complicated for a non-technically experienced person to accomplish. The process of editing an existing Yarn file is no more complicated than replacing that file in the file system. However, you may need someone with Godot experience to test the new script and to deploy new builds.