Navigating Narrative Game Creation

A Beginner's Guide to using Unity & Fungus to Create Narrative 2D Games

By **Linda Ding (2T3) & Livia Nguyen (2T3)** | May 4th, 2023

TABLE OF CONTENTS

Preface

Hello! We (the authors) are creating a video game for our Master's Research Project (MRP) and have thus decided to take on a special topics course to help prepare us for the development of our MRP. This document is a product of said course and was written to document our findings and lessons learnt while creating a prototype of our MRP.

This document is dedicated to future Biomedical Communication students who may be interested in creating educational narrative video games but don't quite know where to start due to limited experience in this area. Specifically, this document will provide insight into why we chose to use the Unity game engine and a plug-in called Fungus to create our narrative 2D game, some of the problems we can across (as well as our solutions) and other helpful tips for getting started.

About Our MRP

Before we dive into why we chose to use Unity or Fungus, it might be helpful to know a bit about the project we wanted to use these programs for.

For our MRP, we are creating *navEDI*, a visual narrative educational 2D video game designed to teach medical students about EDI (equity, diversity and inclusion). The storyline of *navEDI* will revolve around a clinical interaction in an emergency room (ER) that can be played from two different perspectives: 1) the medical student perspective which provides a safe learning environment for practicing clinical skills and 2) the patient perspective which explores stigma, cross-cultural barriers, and empathy through perspective-taking. The game features a point-and-click interactable 2D environment as well as a choose-your-own-adventure mechanic where the user's choices alter the course of the story, resulting in different endings. The primary target of the game is 3rd year medical students from the University of Toronto on clinical psychiatry rotation at the Centre for Addiction and Mental Health (CAMH). *NavEDI* is a downloadable computer game application intended to be used by students outside the classroom/clinical setting. Secondary users of the game include students in adjacent fields and other allied healthcare workers.

These are some of the key features of our game that need to be taken into consideration when deciding how to build the game and what software to use:

- First person perspective & thoughts
    - Player will see through the eyes of the person who's story they are experiencing
    - In-game dialogue text will include inner thoughts from the user's avatar and provide insight into their feelings which may not be outwardly shown
- Explore & interact with the environment and other non-playable characters (NPCs)
    - Certain objects/people in the 2D environment are highlighted as interactable - can click to manipulate or enter dialogue
    - Fixed camera position
- Branching dialogue choices that affect game outcomes
    - When in conversation with key characters, the user may choose between multiple (3-5) dialogue choices - choices will affect outcome of conversation and emotional states of other characters
- Original 2D illustrations for the environments and characters will be incorporated
- Soundscape reflects location & interactions with environment
    - ER sounds will be integrated into the game to add immersion
    - Additional sound effects will be triggered when interacting with certain objects or characters as well as the UI elements

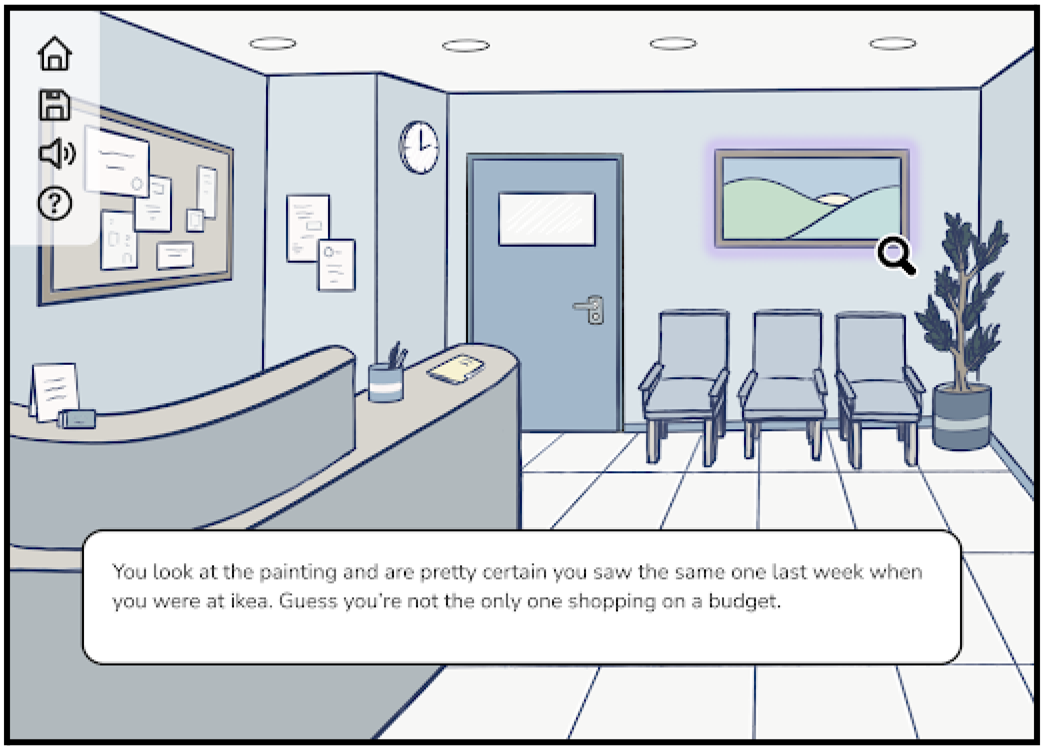

You look at the painting and are pretty certain you saw the same one last week when you were at ikea. Guess you're not the only one shopping on a budget.

Figure 1: An example mock-up of an in-game moment from navEDI. Users can explore and interact with certain objects in the ER to learn more and immerse themselves in the environment.

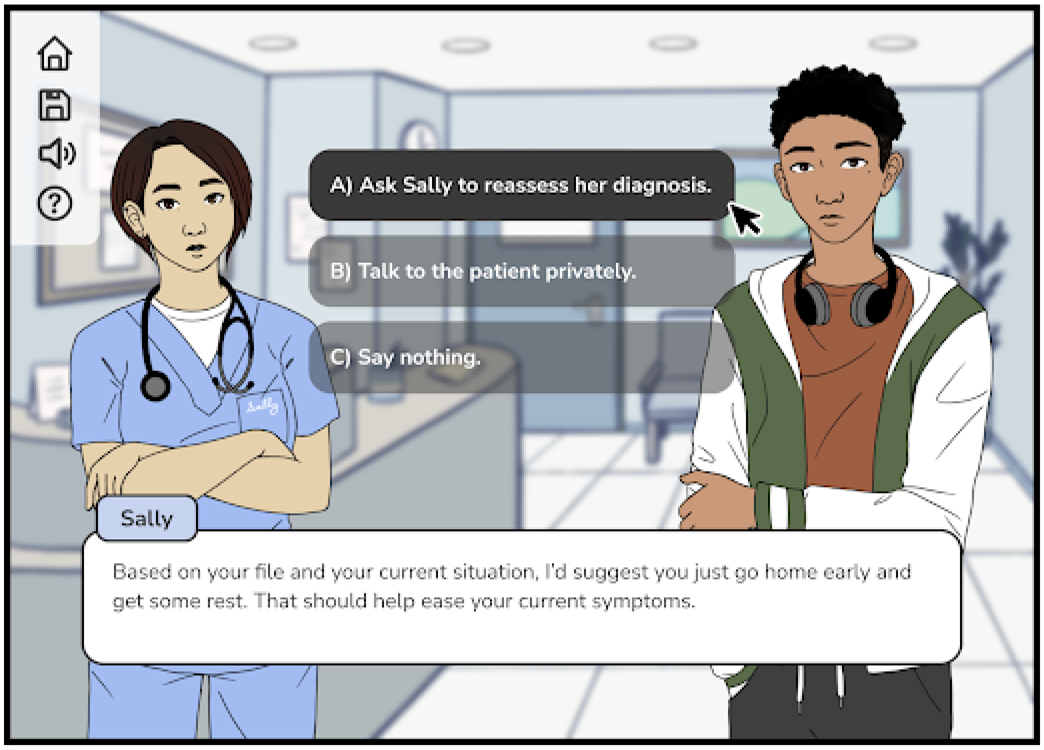

A) Ask Sally to reassess her diagnosis.
B) Talk to the patient privately.
C) Say nothing.
Sally
Based on your file and your current situation, I'd suggest you just go home early and get some rest. That should help ease your current symptoms.

Figure 2: An example mock-up of an in-game moment from navEDI. At key moments, users will be able to choose the next course of action through dialogue choices. These choices will affect subsequent conversations and alter the course of the story.

Choosing a Game Engine

Why We Decided on Using Unity with the Fungus Plug-in to Create our Narrative Video Game

First off you might be thinking, **'What is a game engine?'** A game engine is defined as being a set of software tools or application programming interfaces that optimize the development of a video game. Game engines provide game developers with a framework that enables them to create a video game without having to build/code all the required systems/components (e.g.

physics, graphics, rendering, collision detection, etc) from scratch. Examples of some popular game engines used today include Unity, UnReal Engine, and GameMaker.

Now that you understand the purpose of using a game engine when developing a video game, we'll walk you through our process for deciding on an appropriate game engine for our project. We'll start first with an introduction to some of the different game engines we considered, how they operate, their advantages, limitations, and etc.

Unity

Unity is a 2D/3D game engine and powerful cross-platform IDE (integrated development environment) for developers. It is primarily used to develop video games and simulations for computers, consoles, and mobile devices.

We ultimately chose to use Unity for the development of our project. Unity's capabilities would allow us to create the base visual narrative game in a visual novel-like style with branching dialogue choices and interactable objects in a 2D environment. While working with Unity would be a steeper learning curve, we knew this engine would allow us to work with less limitations and create a game with more interactivity. Since Unity is capable of building a wider variety of games compared to Ren'Py, which is specific to visual novel style games, learning Unity would allow us to develop skills that are more transferable and applicable to potential future projects. Additionally, we knew that the Fungus plug-in for Unity could be used to assist us in building the game by providing us the base code for interactions common to visual novel style games.

Unity	
Language(s)	C#
Cost	Free with student plan
Advantages	- Unity is a cross-platform engine. - Unity editor is supported on Windows, macOS, and Linux platforms - Extensive Unity user manual available - Student plan provides access to many free assets and learning resources (lesson plans for beginner to advanced users)
Limitations	- Requires more in-depth coding knowledge

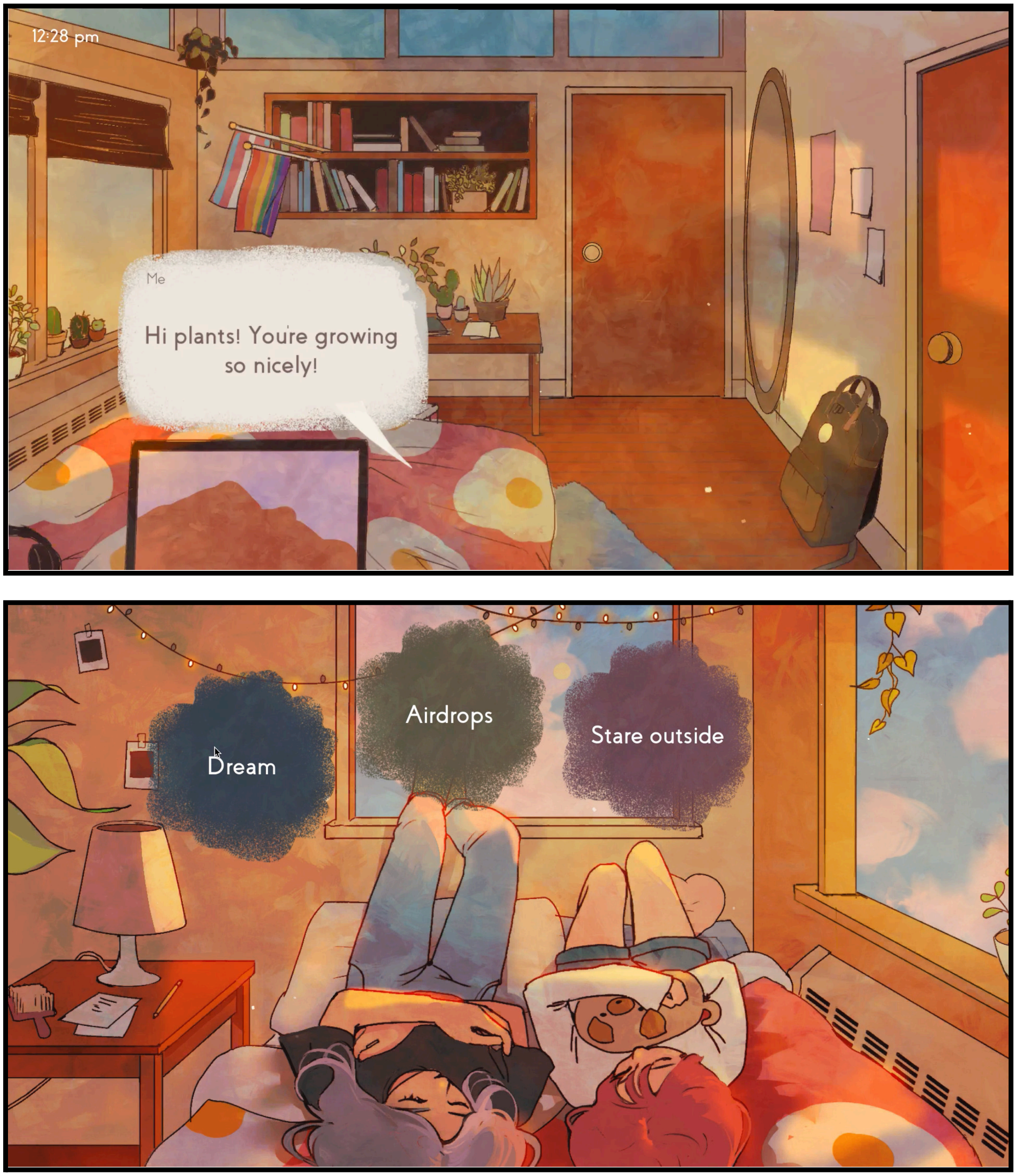
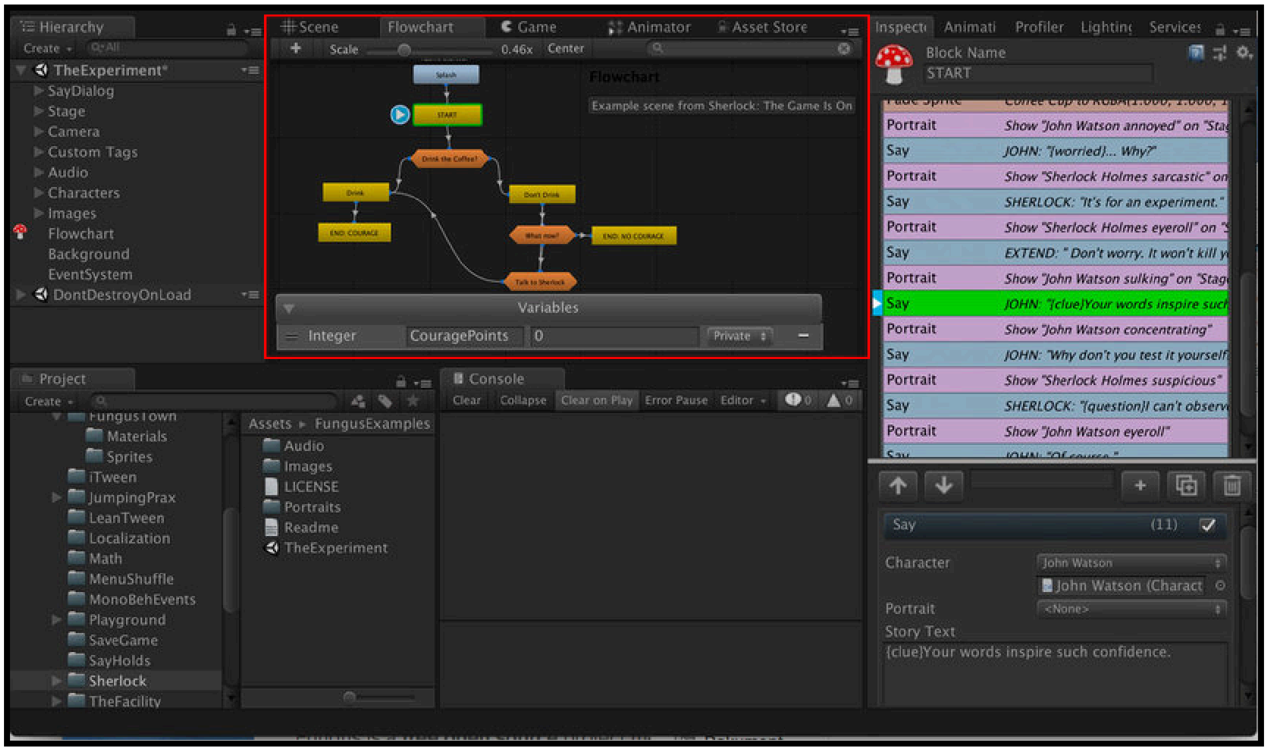


Figure 3: Game moments from [Missed Messages.]() (2019) by [Angela He](). Missed Messages. is a short visual novel game about life struggles and the importance of communication. The game was created in Unity and features dialogue choice options that can lead to 4 different endings.

Fungus

Fungus is a free open-source visual scripting based plug-in for Unity developed by Snozbot. Fungus is made for building narrative games and designed to be easy to learn for beginners to Unity, especially for those with no coding experience. It features an intuitive visual scripting system (workspace and windows) that allows users to add storytelling features (dialogue, sprites, music/sfx, etc.) to their Unity game without requiring one to code.



Portrait	Show "John Watson annoyed" on "Sta
Say	JOHN: "{worried}... Why?"
Portrait	Show "Sherlock Holmes sarcastic" on
Say	SHERLOCK: "It's for an experiment."
Portrait	Show "Sherlock Holmes eyeroll" on "
Say	EXTEND: " Don't worry. It won't kill y
Portrait	Show "John Watson sulking" on "Stag
Say	JOHN: "{clue}Your words inspire such
Portrait	Show "John Watson concentrating"
Say	JOHN: "Why don't you test it yourself
Portrait	Show "Sherlock Holmes suspicious"
Say	SHERLOCK: "{question}I can't observ
Portrait	Show "John Watson eyeroll"

Say (11) ✓

Character John Watson
John Watson (Charact

Portrait <None>

Story Text

{clue}Your words inspire such confidence.

*Figure 4: A screenshot of Fungus's visual coding interface featuring one of its most defining features: the **flowchart** which helps users visually see how different parts of code interact.*

There are many tutorials and courses available for getting started with Fungus and that walk a new user through the process of developing a visual narrative game from beginning to end. Fungus simplifies the coding process by using drag and drop blocks that can be connected together in Fungus' flowchart window while still allowing for customization through C# code.

Fungus has many notable features that lends itself well to creating narrative games. One of these is the **Say Command.** This command is the base building block for dialogue within Fungus and can be programmed to be associated with a specific character in your story. This means that whenever a specific character speaks in-game, their dialogue is associated with a certain colour, style and sound effect. Fungus also has **Portraits** which can be linked to these Say Commands so that an image of the character speaking appears with their dialogue text.

Portraits can be animated and you can have multiple Portraits for one character. This makes it easy to swap different versions of character art (e.g. a character with multiple expressions). **Menu commands** allow you to easily create a branching narrative through through the presentation of dialogue choices to the player that link to other game events.

Many game studios use Fungus mostly for its narrative system, but there are many tutorials available that explain how to use Fungus to create games beyond this (e.g. turn-based role-playing games)!

Fungus	
Language(s)	Visual drag/drop in flowchart window w/ C# or Lua for customization
Cost	Free and open-source
Advantages	- More feasible than working entirely with only Unity - Easy to use and approachable editor window for setting up interactions and more - Documentation and training videos available on website - Decently active community and discord channel
Limitations	- Still requires coding for customization - Can make it difficult to add/modify certain game features (e.g. adding animated portraits/expressions)

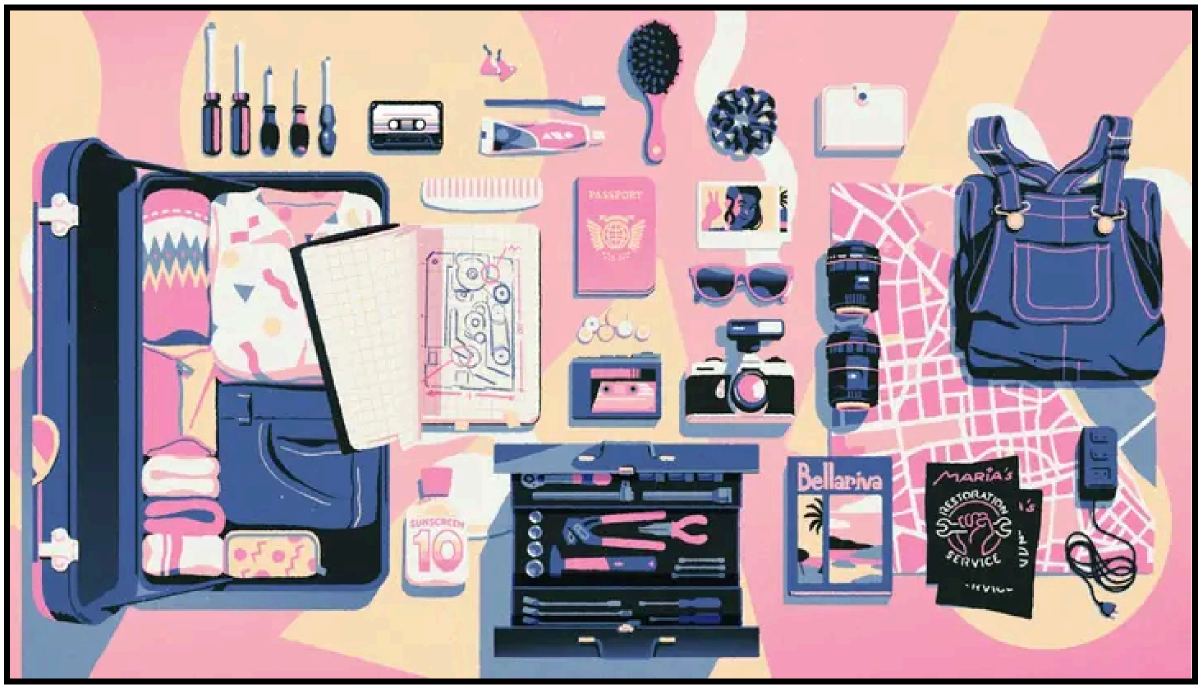


Figure 5: A screenshot of a BAFTA-nominated narrative puzzle game called [Assemble with Care](#) (2019) made by Ustwo Games using Fungus. Check out this [article](#) to see how they used Fungus for their prototypes!

Alternative Options

There are many game engines available out there. Therefore, it's important to consider some of the following questions when choosing a suitable engine for your project:

- What type of game do you want to create (2D/3D)?
- What are the main features you want your game to have (mostly dialogue-based, etc. point-and-click exploration, etc.)?
- What is your comfort level or experience with coding in various languages?

The following game engines can also be considered for creating a narrative game.

RenPy

Ren'Py is a popular game engine that can be used to create visual novels and life simulation games that can operate on both computers and mobile devices.

We were first introduced to Ren'Py through Willow Yang (2T1) who used the engine to develop their MRP, [Adventure Down Hidden Depths (AdventDHD)](#). AdventDHD is a gamified self-directed ADHD coaching tool that uses the visual novel medium to create a unique and engaging e-learning experience.

We considered Ren'Py as our Plan B game engine for development due to its specific features for creating narrative games, intensive documentation, active community, and our familiarity with the Python language.

Ren'Py	
Language(s)	Python
Cost	Open source and free
Advantages	- Anyone can create a visual novel without needing to code so it's much easier to pick up than Unity - Python scripting allows for customization and the creation of more complex simulation games - Extensive documentation (comes with detailed reference manual) - Many tutorials and plug-ins available - Popular game engine so decent community
Limitations	- Limited movement of character in 3D so would be limited to mostly point/click interactions with the environment

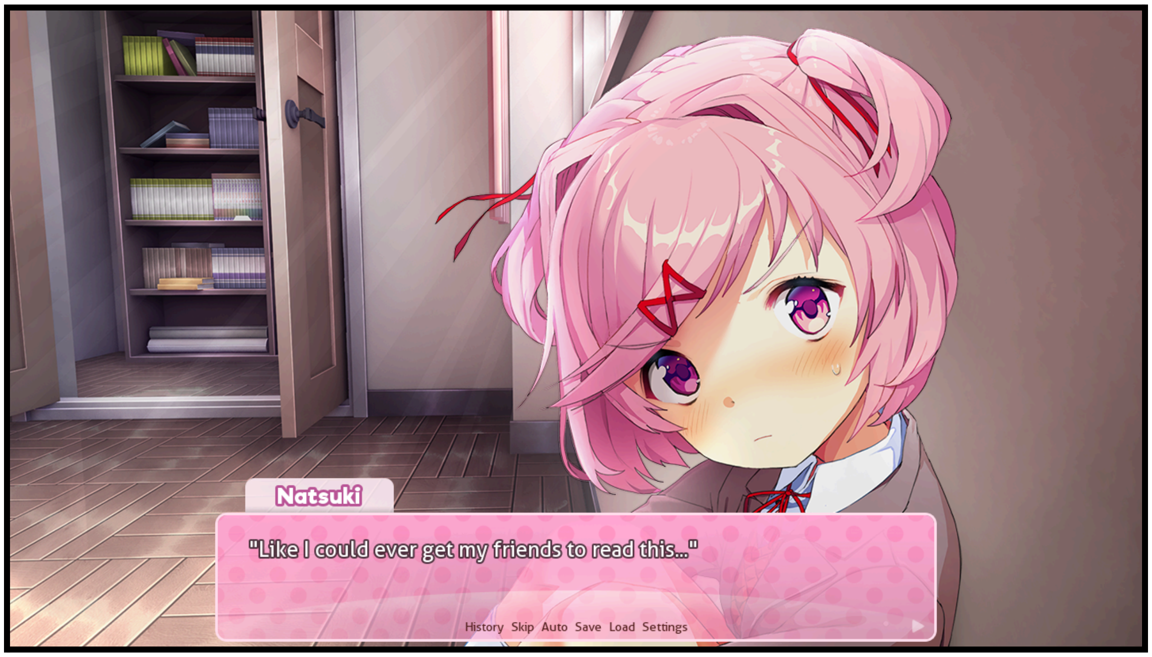

Natsuki
"Like I could ever get my friends to read this..."
History Skip Auto Save Load Settings

Figure 6: A screenshot of a popular visual novel game called Doki Doki Literature Club (2017) made by Team Salvato using Ren'Py.

Twine

Twine is a free, open-source tool that can be used to create interactive non-linear or branching narrative experiences through hypertext. You can either download the desktop application or just use it in your browser. A good place to start learning Twine is its documentation called the Twine Cookbook. To get the most out of Twine, it is helpful to have some knowledge of HTML, CSS, or JavaScript. Nevertheless, even without extensive coding you can add text-based designs, 2D pictures, and other simple audio/visual elements. As our MRP script features a branching narrative where in the player follows an overall storyline but is able to make choices that affect individual sections of the plot and gameplay, we decided to use Twine to create an interactive prototype of our draft scripts so we could collect feedback.

Twine	
Language(s)	HTML, CSS, or JavaScript
Cost	Free
Advantages	- Can be used to prototype large branching story games to test the progression of choices and visualize different player routes - Outputs to an HTML file that can be sent and opened on any computer - Possible to integrate many different features through code (e.g. arrays, audio, conditional statements, date and time, sidebars, images, keyboard events, loading screen, pop-up windows, player statistics, saving games, etc.) - Documentation is well-structured and includes a number of helpful examples
Limitations	- Not entirely intuitive, quite a bit of coding required for more complicated branching and game-like features - Cannot build the whole game in Twine to include interactive components - Community is not as big (although still quite good) and it's more difficult to find answers to your questions on forums

Twine Basics

The Twine editor calls individual projects "**stories**". The main components of Twine stories are "**passages**". Passages are basically blocks of dialogue or sections of code. You can connect passages by adding two opening and closing brackets around the title of another passage. In the compiled HTML version of the story, there will now be a link to navigate between the two passages. This is how branching choices are created.

Story Formats
Before starting anything in Twine, you first need to decide on what type of format your Twine story will take as there are multiple options. Each story format provides a different visual layout, set of macros (functions), and internal JavaScript functionality. We recommend looking through the different formats and finding the one that works the best for your abilities and what you want to create.

Based on our experience with two of the Twine formats, this is what we can say about them...

- **Harlowe** is the default story format for Twine and is good for beginners as it is easy to learn and has a bunch of built-in features as well as more of a detailed editor interface.
- **SugarCube** (which is the format we chose to use for our script prototype) has more functionality than Harlowe but it helps to have some experience with HTML, CSS, and JavaScript to fully make use of this story format.

Twine Tips!

- ❖ Before you start creating anything in Twine, you should have a **script** to work with - it doesn't need to be final as it is still relatively easy to make changes later down the line, but it will help a lot with planning/linking passages as you work.
- ❖ If you have no previous experience in Twine, start with a **small practice story** first - something simple with a couple of choices just so you get a hang of things.
- ❖ Keep in mind the following consideration for creating branching narratives:
    - ➢ Does a character or interaction serve a **specific purpose** or is it filler? The consequences of the player's actions should be visible through **meaningful changes** to the game narrative and/or mechanics. It’s important to respect the player's agency and honour their choices.
- ❖ There are many cool **Twine templates** that are free to download on itch.io and Github! Here is an example of one: https://awmorgan.itch.io/twine-sugarcube-template

Getting Started

Preparation

Before you even start thinking about working in Fungus or Unity, you need to have a solid idea for your game. Flush out things like your intended audience, core game mechanics, visual style, user flow, and etc. before starting any development of your game. Consider creating a game design document (GDD) to help solidify the direction you want your game to take. You can take a look at this template to start! GDD's are also useful if you need to pitch your game idea later on. Check out these examples of GDD's: Bioshock and Silent Hill 2.

Now that you have solidified the idea for your game, you can start working on a script for the dialogue portion of the game as well as some basic visual assets (e.g. character art). The assets don't need to be finalized at this point but can act as a placeholder as you start to play around in Unity. We recommend giving this article by Angela He a read! It gives a good overview of all the things you need to consider as you make your game from start to finish.

Getting Started Tips!

- There is a lot to learn in Unity so **start as soon as you can!**
  - We (the authors) started learning Unity and exploring narrative game development during the summer between 1st and 2nd year.
    - This helped us a lot because during the term, we were super busy with course work and getting a head start over the summer relieved some pressure.
- **Get inspired!**
  - Unity has a bunch of case studies that explore how various games use the platform:
    - https://unity.com/case-study
  - Check out game development logs / blogs.
    - There are some game devs that stream their progress on Twitch or upload progress videos on YouTube.

        - Check out Game Developers Conference videos (our recommended watches for narrative games are located in this list in the Resources section of this document).
- **Play video games!** (for research purposes only of course)
    - There are so many amazing narrative games out there that you can find inspiration from!
        - You can find games on video game publishing platforms such as Steam (https://store.steampowered.com/) or itch.io (https://itch.io/).
        - You can also refer to our list of curated narrative-focused video games that we recommend you play.
    - It is important to not just passively play these games. Take note of things that you think worked well and didn't work as well (within the story, UI, art, sound, gameplay, general game experience, etc.).
        - We organised our findings on a Miro board and decided on key takeaways that we wanted to consider for our game.

Downloading Unity & Fungus

Both Unity and the Fungus plug-in are free to download and work on both Windows and Mac OS. See the instructions and links below to get started...

Unity
To download Unity, go to this link https://unity.com/ and choose a plan. The personal plan is free or you can apply for the student plan through this link. You will also need to download Unity Hub, which you will use to manage multiple installations of various versions of the Unity Editor, create new projects, and access your work. For our prototype, we used the newest version of Unity (2021.3.24f1) as this was recommended in the Fungus installation instructions.

Fungus
Fungus is not currently available in the Unity Asset store (even though the main Fungus website suggests that you can get the plug-in from the store). Instead, you must download Fungus from Github through the following link: https://github.com/snozbot/fungus/releases. At the time of writing of this document, we used the most recent version of Fungus: v3.13.8. Once the package is downloaded, import Fungus into your Unity project by selecting Assets > Import Package > Custom Package > navigate to downloaded fungus package and open > in the

popup box, select “All” and then “Import”. If there is a warning about changed API and whether to automatically update, choose “Just for these files” for now.

Setting up a Git Repository

What is Git?

Git is a version control software that can be used to manage contributions to a project within a team as well as track where changes to the project have been made over time. If someone changes a file, Git records the differences between the old and updated version and maintains a history. This allows one to preserve different versions, go back to a previous version (version control history), and review changes.

GitHub is a public platform to host a git repository online. It allows for easier collaboration with multiple people as you can see all the files, their source code, and what the contents of each commit is. GitHub supports branching which allows people to develop the same code separately without interfering with one another. Git will be able to determine when there is conflict between the branches and can help merge the changes. On GitHub, if you make your project “public" (as opposed to private), people can view your code, learn from them, help modify and work on parts of it, etc.

There are many free resources online that explain the basics of Git and related terminology you should know when working in Git. Here is an article and a link to the official Git documentation that you can check out!

How Git can be useful for your project

Setting up a git repository for your project allows for smoother collaboration between developers as you don’t have to pass files back and forth and collaborators can work on different sections of the project simultaneously. Overall, this makes development more efficient, stream-lined, and safer as you can work with a history of all the changes made. We highly recommend setting up a Git repository, especially if you are collaborating with multiple people on your project!

How to set up a Git Repo

- To create the git repo for the Unity project, there are many resources online. Here are some that you may find helpful:
    - This YouTube video by Brackeys: How to use GitHub with Unity

        - If you want to make use of Git Desktop and the Github Browser (recommended)
    - This article by Unity at Scale:
        - If you want to create the repository through using a command line window or terminal (this method is not recommended if you have no previous experience working with this window/terminal)

Git Tips!

- When creating your repository, make sure to name it appropriately, add a brief description, and choose if the repo is private or public
    - Generally, if you plan on using paid 3rd party asset files, you should not be committing these to a public repository (effectively giving away these assets for free). It might be safer to go with a private repository.
- Make sure your repository has a Unity **.gitignore** file.
    - This is a text file that tells git what files should not be included in the version control - ultimately, this will save space and reduce the size of the uploads to the repo by omitting some of the temporary meta files Unity creates that are only used by Unity and aren't necessary to be included.
- If you are collaborating with others and someone else has committed changes to the repo, fetch and pull first on your end before adding onto the project and continuing to make changes.
- Leave **useful descriptions** when you commit your changes!
    - Include things like what was changed, what still needs to be fixed, any other important things to note
- Check out this free Udemy course on Git.

Learning Unity & Fungus

A look into our mindset for learning and some basics in Unity and Fungus (just enough so you can hopefully understand the rest of this document) as well as some tips!

Introduction

It’s important to have an open mindset when starting to learn Unity. Try not to be discouraged easily! There is a lot of learning to do - you will definitely have to learn as you go and it’s not possible to know everything before starting a project in Unity. Just take your time and try to have fun with it!

We highly recommend building some sort of foundation in the basics of coding first as this will help you greatly later on. It is risky and unsustainable to just rely on copying code found on forums and trying to build out a full project this way. If you take the time to build up some type of foundation, you will be able to understand what you are looking at in the code and how it can be applied to your projects. This also opens up the possibility of tweaking any code you find to suit your own needs!

Where to Begin?

A good place to start learning Unity is the Learning Pathways on the Unity Learn website which are guided learning experiences created by Unity. You can track your progress as you progress through the structured lessons, earn badges, and try out practice exercises. A word of advice: be selective on which lessons you spend the most time on. So for example, if you plan on creating a 2D game, perhaps you can spend less time on the 3D-focused tutorials. Refer to the Resources List to see what tutorials we focused on in the Unity Learning Pathways.

To start learning some Unity basics, you can also check out the “Working in Unity” Section within the Unity Manual. It walks you through the basics of Unity’s interface, working with Scenes, GameObjects, asset workflow, etc. There are also numerous tutorials available on YouTube (Brackey’s channel has a ton of helpful tutorials) for learning Unity.

If you want to start building a foundation in coding with C#, there are also an astounding amount of free resources available online such as Microsoft’s free courses on C# or W3School’s learning resources. You can also give this Unity article on learning C# for beginners a read. Finally, we encourage you to make your own notes as you embark on your Unity/Fungus learning journey as you can refer to these notes later once you start working on your game!

Where to Find Help

Both Unity and Fungus have their own dedicated documentation. You should be referring to these resources constantly as you build out your game.

Unity

- Unity Community
    - Forums, blogs, etc.
- Unity Documentation
    - Unity Asset Store
        - Contains free and commercial assets made by Unity and members of the community - lots of free hidden gems
        - Can import and download them directly in the package Manager window in Unity
    - Unity Editor Manual
        - A user manual to Unity that should be consulted frequently
    - Unity Scripting Reference
        - Useful to have open when scripting
            - Describes methods and properties of various classes with example code

Fungus

- Fungus Documentation
- Fungus Tutorial Videos
    - Some may be outdated, but still gives good overview
- Link to join the discord server for fungus:
    - https://discord.gg/99RqraQ
        - Quite active! People are friendly and helpful!
- Fungus Forums
    - Not as active as the discord server but there still seems to be one dedicated admin who moderates the forums and replies to questions.

Basics of Unity

Unity Editor Interface

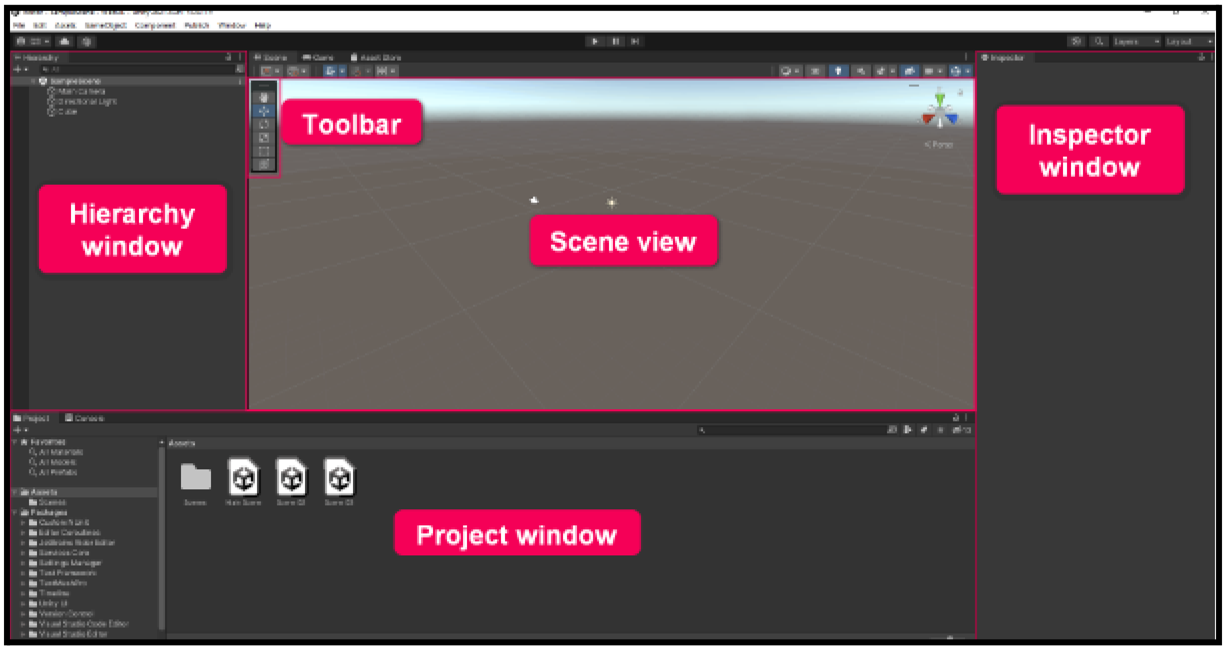


Figure 7: A screenshot of the basic Unity interface.

Scene view / Game view

- Interactive window to edit the current Scene and view the game world
- Press to enter Game view to test game live in the Editor
- Note: Important to stop the game completely if making permanent changes
- Can select and manipulate objects

Hierarchy window

- List of GameObjects in Scene and their parent-child relationships (similar to the layer window in Adobe Photoshop for example)

Inspector window

- Displays components/settings of selected game objects, each component contains properties/behaviours of the game objects that give it functionality

Project window

- Acts like a file explorer - displays and can organize assets/files for project
- Can add folders and create new game objects by right-clicking in this window

Key Concepts

Unity projects are made up of assets and scenes along with other components. A **scene** is a fundamental unit of the playable world in Unity. A project must have at least one scene but your game can be organized into multiple scenes that can be considered discrete experiences. For example, you can have one scene per game level, a separate scene for the main menu, etc.

These scenes contains **GameObjects** (GO) which are made of **components** that describe the properties and behaviours of each GO. The transform component is automatically attached to a GO but you need to add others manually through the inspector.

The main camera in the hierarchy is the camera that captures and displays your Scene to the player as in Game view. You can move the camera using transform tools and also switch between 2D/3D views. Each camera is equipped with an audio listener component that will be used to "listen" to audio added to the scene.

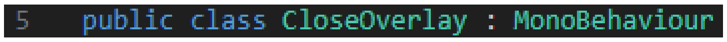
Tips for Working with Unity!

- Avoid **feature creep** which is the tempting but risky tendency to add shiny new features to a product vs. sticking to the ones agreed upon during pre-production.
- Keep your Unity **hierarchy and files organized**! Make sure each asset, script, etc. is named appropriately and organized into appropriate groups/folders.
- **Adding audio** (background music, sound effects, etc.) to your game is really the cherry on top. Trust. It makes such a big difference.
  - Sound can create mood (establishes how the audience should feel about what is occurring), a sense of space (provides a sense of where they are, continues to provide context), enhance interactions (suggest interactions between an object and an environment), and more.

Scripting in Unity

Unity uses the **C#** programming language. Here is an article on best practices when coding in Unity. Scripts that you create can be applied as a component to a GameObject and linked to different buttons to tell Unity what that button should do when clicked for example.

To start a new script:

- Assets → Open C# Project → opens Visual Studio (Unity's default editor) OR Create → C# Script → open to edit
    - Note: Make sure the name of the script is correctly referred to in the MonoBehaviour class within every script. For example, if my script is named "CloseOverlay", at the beginning of my script, there should be a line that looks like this: 

      ```
      public class CloseOverlay : MonoBehaviour
      ```

      .
        - If the name of the script does not match this, you will get an error!
- When you create a new script in Unity, the default file will contain code that imports libraries needed to run the script and 2 basic functions: Start (called once when the program first runs) and Update (called once per frame).

A recommended general workflow for scripting:

- Determine the main interactions and game mechanics you want for your game. Organize these based on priority and take into consideration their feasibility given your project timeline.
    - Consider using multiple KanBan charts to keep track of the progress of each feature as it is coded into your game.
- Break these down into more granular parts/tasks that can be translated into chunks to eventually code.
    - For example, if your goal is to move a GO a certain amount forward when the user clicks the button. You can break this into small tasks such as finding the initial position of the GO, storing this position as a variable, moving a GO by a certain amount, etc.
- Experiment with completing these small tasks using basic assets in a separate scene to test your idea. Gradually, build up the tasks to code the full interaction.
    - Make sure to check that each line is working properly and that you understand what each line of the code does.
- Finally, integrate the knowledge you have gained and code you have developed in these smaller test scenes and integrate them into your main Unity project.

Tips for Scripting in Unity!

❖ It's important to do **beta tests** well in advance of the deadline to ensure you have extra time to solve any bugs as you will likely come across lots.

- Add **useful comments** throughout your scripts. These comments should be detailed and frequent enough so that someone can understand your code.
- **Don't give up!**
  - It's super rewarding when you figure stuff out on your own after trying your best to push through and problem solve - you will learn a lot more this way as opposed to googling everything when it starts to get slightly hard.
- At the same time, knowing **when and *HOW* to ask for help**
  - Sometimes, some things you are trying to accomplish are outside of your current skills/knowledge and this is perfectly fine!
  - There are many ways you can reach out for help such as Unity's forums, the Fungus discord, friends who code, professors who are knowledgeable in Unity. A big part is also knowing how to format/ask your question so you get a good answer.
    - Check out this forum post for some great tips!
    - If you posted a question to a forum and end up finding a solution, make sure to post it! It can help someone in the future who has the same question as you.

Basics of Fungus

A good place to start is the Getting Started section of the Fungus wiki.

The **Flowchart** is one of the main areas you will be working within when using Fungus. This window displays all the **blocks** (essentially groups of code) and visualizes how each of the blocks are connected to each other through a flowchart. Within each block are Fungus **commands** which can be added through the inspector of the block and link blocks together. If you are not as familiar with how Unity UI works, it might be a bit confusing on how to change the default settings. **Event handlers** determine when a block (and its associated commands) should be executed.

To display the Flowchart editor window, select Window > Tools > Fungus > Flowchart Window. For convenience, you should dock the Flowchart window somewhere so it is easily accessible. You can have multiple flowcharts in one scene although you must make sure only one has Execute On Event: Game Started block or there will be conflict.

There are 3 types of blocks:

1. Event (blue rounded rectangle)
    a. These blocks execute on an event (e.g., the Game Started Block which is the first block that will be executed Automatically created in flowchart)
2. Branching (orange long hexagon)
    a. Calls 2 or more other blocks. Arrows show flow of execution.
3. Standard (yellow rectangle)
    a. Doesn't branch to 2 or more blocks

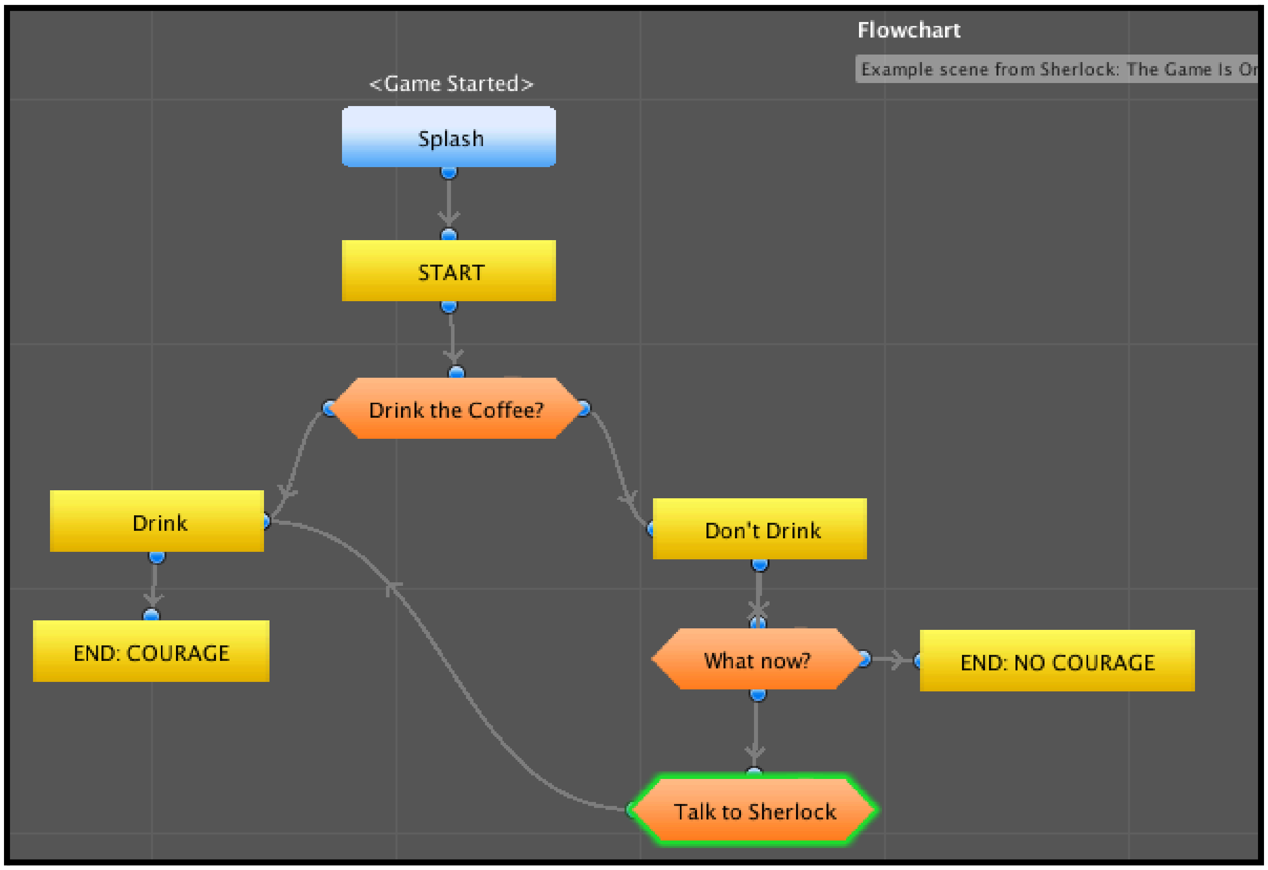


Figure 8: A screenshot of a Fungus Flowchart with the 3 types of blocks. This flowchart is from an example scene included with the Fungus package.

First Steps in Fungus

- Play through some example Fungus scenes
    - In the project window, open (for example) The Hunter scene (FungusExamples > TheHunter > TheHunter.unity).
        - Press play and click through the example game.

    - Try playing the other example scenes and watch how the gameplay is controlled by the flowchart in the Fungus Script window.
- Start playing around in Fungus or find some tutorials to follow! Here is a good video to start off with by JollyFranchers!

Tips for Fungus!

- ❖ Have a refined script before starting anything in Fungus.
    - ➢ This will make it easier to work in Unity and create the Fungus flowchart as it can take up valuable time if you need to go back and make lots of small changes to your script once you have started coding.
- ❖ Make use of the debug command script that comes with Fungus!
    - ➢ You can add this to any block, input a variable or component you want to check and the command will send a message to your debug console. This script can be found under Fungus > Assets > Fungus > Scripts > Commands. DebugLog.cs

---

Some Struggles & Solutions

Some examples of issues we encountered while working with Unity and Fungus & how we solved them. Hopefully this can help you if you ever encounter a similar problem!

Case 1: Using Player-Generated Text Input

In video games, sometimes there is a screen were the player can create their own character. At the end of this character creation screen, the player is expected to name their character which is then used in dialogue throughout the rest of the game. This requires the game to collect and utilize a player-generated text input.

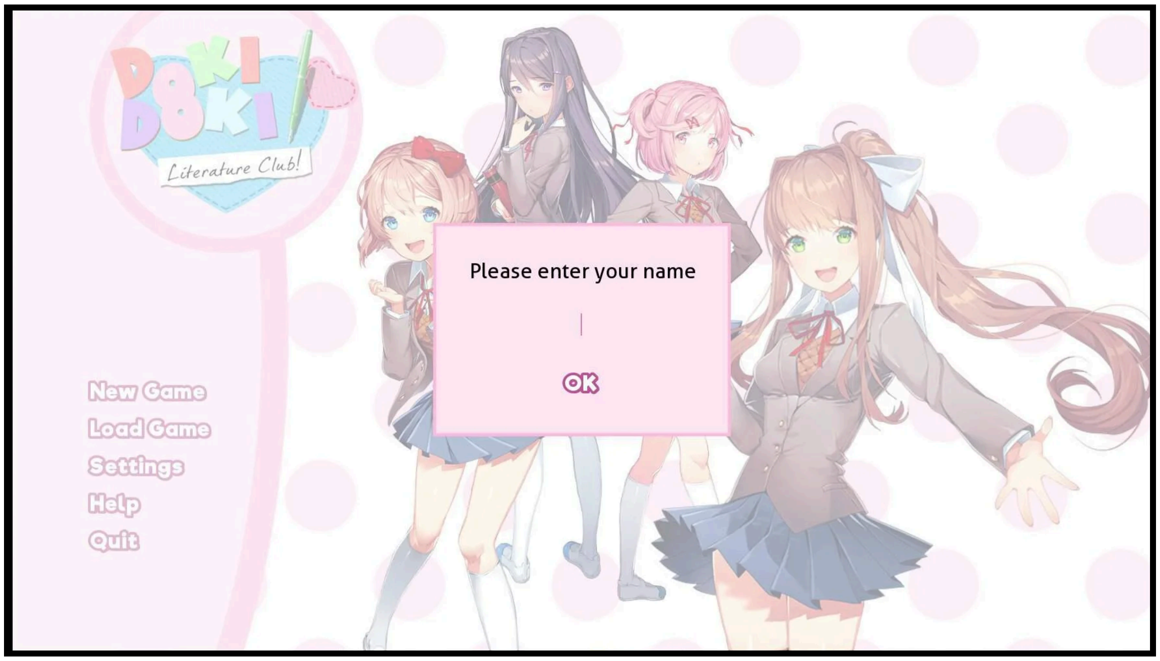


Figure 9: An example screenshot from *Doki Doki Literature Club* where the player is asked to enter their name as a text input. This name is then referred to later in the game.

Problem:
As someone new to Fungus and Unity, it may be unclear how to do this at first and once you start to look into it more, you may realize that you need to use variables and set up a variable and various things in the UI and within the components for this to work. There are scattered answers on how to do this in various forums as well as some tutorial videos.

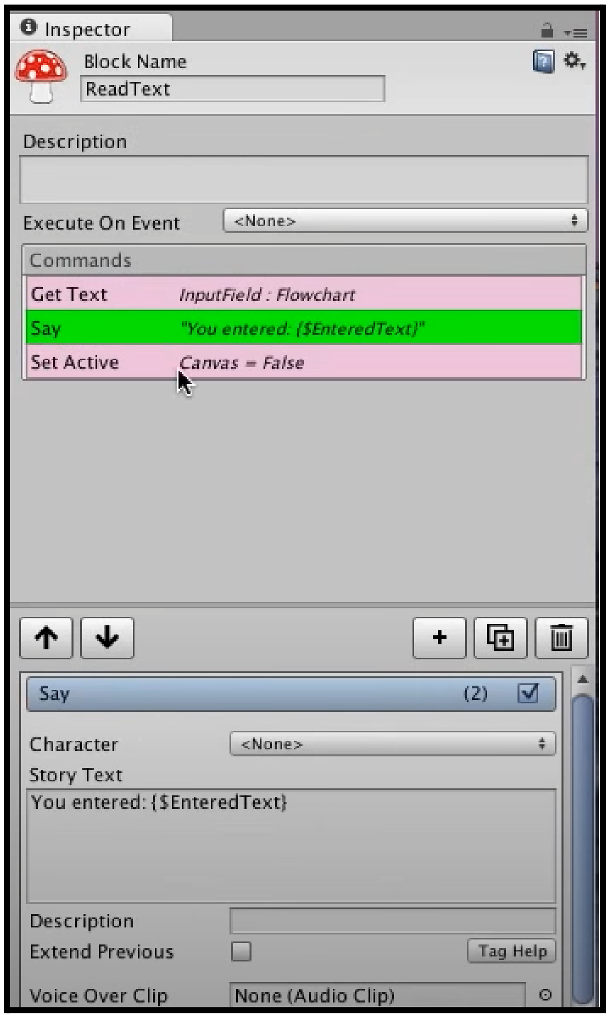
Solution:
There are a number of steps to set up the appropriate commands within a flowchart block. Note: The following steps are for using a user-generated name but these same principles/concepts for collecting input and storing it in a variable can be applied to other cases.

Steps:

1. Add an input field to your canvas though GameObject > UI > Panel, Button, Input Field.
2. Create a flowchart block within the flowchart window:
    a. In “Execute on Event”, select End Edit.
    b. Put the input field you just created into the Target Input Field component.
        i. Once the player presses enter, the flowchart block will enter.
3. Determine your string (text) variable that will store the inputted text. For example, “playerName” if the text input will become the player’s name.
    a. This variable should be formatted like {$playerName}

   b. Make sure the variable you pick up the name in is set to Public (not Private) so it can be used in other flowcharts within the scene.

4. In the flowchart with end edit, add a Get Text command. To set up this command, place the text box from the input field in Target Text Object, and select the String Variable you want to hold that information.
5. In any block following this, add a Say command and type out the dialogue text as usual including the variable.
   a. This say command should now print out the dialogue including the player's name that the player input earlier.



Block Name: ReadText

Description

Execute On Event: <None>

Commands

Get Text	*InputField : Flowchart*
Say	*"You entered: {$EnteredText}"*
Set Active	*Canvas = False*

Say (2)

Character: <None>

Story Text: You entered: {$EnteredText}

Description

Extend Previous

Tag Help

Voice Over Clip: None (Audio Clip)

Figure 10: A screenshot of the commands within a block that is set up to collect the player's text input and store it in a variable to use later on in a Say command.

Resource(s) Used:

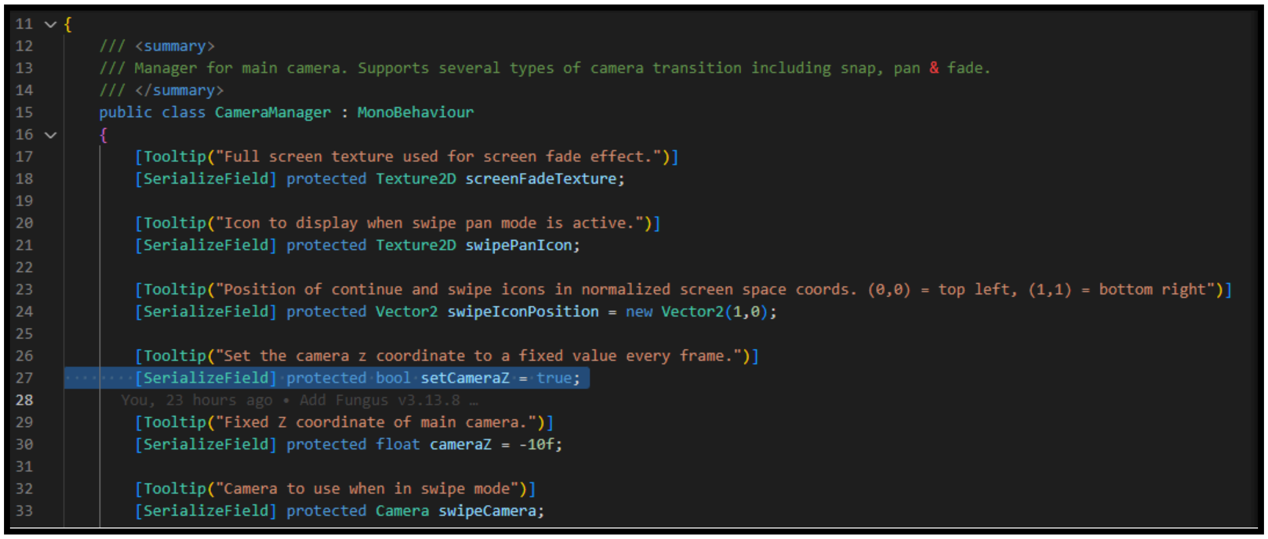
- Fungus 2.2 features
- https://muut.com/i/fungus/general:input-field-help
- https://muut.com/i/fungus/general:adding-enter-player-name
- https://fungusgames.com/forum#!/general:could-someone-help-me-need
- There is an example scene under Assets>FungusExamples>EnterName that you can see in action and take a look at the script included for reference.
- Check out this add-on by Just Us Games which works with user-defined gender

Case 2: Moving the Fungus Camera in a 3D Space

Although our game will be mostly in 2D, we were thinking of ways we can utilise the 3D space that the 2D assets are placed in within Unity to our advantage. We wanted to experiment with zooming in/out or moving the camera in the Z direction during specific moments of our game.

Problem:
The Fungus plug-in is described on its website as a plugin that works for Unity 3D so one may think that the default Fungus camera would be able to move in 3D space freely. Wrong! The current default View system in fungus does not work for 3D environments as the Fungus camera as-is is only able to move in the X and Y directions, and is locked in the Z-direction.

Solution:
Knowing that Unity cameras are able to move in the X,Y and Z direction normally, there must be something different about the Fungus camera - and this is likely through the code it is linked to. So, looking through the scripts for the one linked to the Fungus camera, there was a script named CameraManager.cs within the Fungus folder which seemed promising. Looking through the code, there were a couple lines of code (26-30) that seemed to deal with the Z coordinate of the camera.



```
11  {
12      /// <summary>
13      /// Manager for main camera. Supports several types of camera transition including snap, pan & fade.
14      /// </summary>
15      public class CameraManager : MonoBehaviour
16      {
17          [Tooltip("Full screen texture used for screen fade effect.")]
18          [SerializeField] protected Texture2D screenFadeTexture;
19
20          [Tooltip("Icon to display when swipe pan mode is active.")]
21          [SerializeField] protected Texture2D swipePanIcon;
22
23          [Tooltip("Position of continue and swipe icons in normalized screen space coords. (0,0) = top left, (1,1) = bottom right")]
24          [SerializeField] protected Vector2 swipeIconPosition = new Vector2(1,0);
25
26          [Tooltip("Set the camera z coordinate to a fixed value every frame.")]
27          [SerializeField] protected bool setCameraZ = true;
28          You, 23 hours ago • Add Fungus v3.13.8 …
29          [Tooltip("Fixed Z coordinate of main camera.")]
30          [SerializeField] protected float cameraZ = -10f;
31
32          [Tooltip("Camera to use when in swipe mode")]
33          [SerializeField] protected Camera swipeCamera;
```

Figure 11: A screenshot of the CameraManager.cs script and the lines of interest that deal with the camera's Z coordinate. Line 27 is highlighted.

Since this script is a core component of Fungus, we were a bit hesitant on editing the code without knowing more, as we were unsure if this would affect other things. So after some searching on the internet, we found a forum post that talked about this part of the script and confirmed our suspicions that these lines of codes would have to be changed. Based on this forum post, here are the steps to allow the Fungus camera to move in the Z direction:

1. Open up CameraManager.cs script found under Fungus>Scripts>Components
2. On line 27 of the code, that reads " [SerializeField] protected bool setCameraZ = true;"
    a. Change the bool "true" to "**false**"
3. Save your changes to the script.
4. Note: If you ever update fungus, you will need to go back and make this change again.

Resource(s) Used:

- 04 Fungus Camera
- Confirmation of answer found on the Fungus forum: https://fungusgames.com/forum#!/general:camera-in-3d

Case 3: Dynamic Mouse Cursor

A key part of the exploration part of our game is that the mouse cursor is dynamic and will hint at interactable elements within the environment. For example, a neutral mouse course will be the default visible cursor used for general clicking with the menu screens and for choice selection. There is also a speech bubble cursor that indicates the ability to converse with certain characters within the environment. The last type of mouse cursor looks like a magnifying glass and indicates the ability to observe something more closely within the environment. These last two types of cursors should appear when hovering over the associated object in the environment that they are linked with. This is to give the player a hint as to what type of interaction will happen when they click on the object they are hovering over.

Problem:
It was difficult to find documentation on the Set Mouse Cursor command or to know this command existed without being introduced to it previously as it is nested under Sprite Commands within the documentation. Also, to fully recreate the hover effect envisioned for our game, this command would have to be paired with another edit within the interactable sprite's components itself.

Solution:
Here, we will document what we know about altering the mouse cursor in Fungus and how to create a dynamic mouse cursor as described above.

<u>Set Mouse Cursor Command</u>

- The Set Mouse Cursor command changes the default sprite of the cursor.
    - It can be found under the Sprite > Set Mouse Cursor.
    - This command needs to be applied to the Game Started block (the very first block of the game in the flowchart).
    - To set up the command, you must input the image of your new cursor in the "Cursor Texture" input field.
- An example of use of this command is hidden in the DragAndDrop Fungus Example scene so you can take a look at this for reference as well.

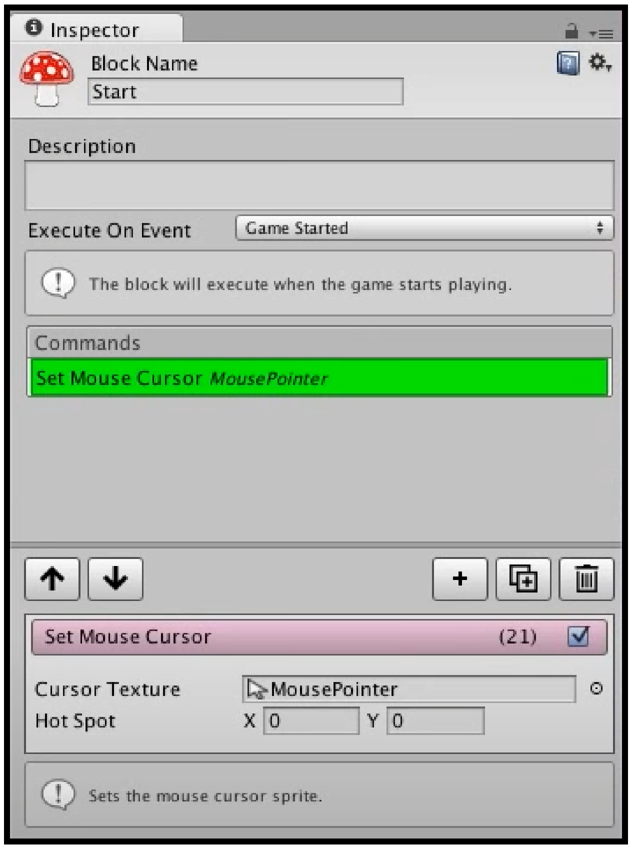


Figure 12: A screenshot of the Set Mouse Cursor command set up in the Inspector. Note that this command executes on the event of the game starting.

<u>Change Cursor On Hover</u>

You can make the mouse cursor change when hovering over interactable objects. This can be done within the inspector of a clickable sprite Game Object.

- To create a clickable sprite GO, Tools > Fungus > Create > Clickable Sprite
- Within the inspector of this GO, drag your hover cursor image into the “Hover Cursor” input field - this field will be empty by default

NOTE: To input your mouse cursor image properly, once uploaded into your asset folder, you must click on the image and change the Texture Type to Cursor.

Resource(s) Used:

- Fungus 2.2 features
- Check out this [forum post]() if you are interested in adding a highlighting hover interaction to sprites.

Case 4: Slot-Based Save System

A **save system** in a video game allows the player to save their current game progress so that they can continue from where they left off at a later time. A **slot-based save system** makes use of “save slots” that the player chooses to save their game data to - this allows you to have multiple playthroughs of the game at the same time or even if set up properly, allows the player to go back to a previous part of their playthrough. There are usually a limited number of slots available and managing these slots (e.g. deleting a save) is done through the game’s menus.

In our game, a save system is an important part of our game’s functionality as considering the length of the game and our end users, we do not expect our users to finish the game in one go. Thus, being able to save their current game progress and return back to the game at the spot they left off at is an important feature for us to include. Also, we intend for the players to have multiple playthroughs of our game so having multiple save slots available is ideal.

Problem:
Unfortunately, the current Fungus Save System is currently in Beta and has yet to be updated for a while now. The Fungus Save System as-is is not a slot-based save system and includes functionality that we did not want in our game such a rewinding and restarting. Also, the current system relies on Save Points which are stored as the player progresses the game to build a Save History. These Save Points must be manually added in the Flowchart at specific points in the game and when reloading the game, the player will be brought back to this specific save point. This system did not exactly allow the player to save their progress at any moment of the game, but just at specific points. Building a new save system from scratch that worked with Fungus or adapting the current one to suit our needs was a task outside of our current knowledge with C# and scripting.

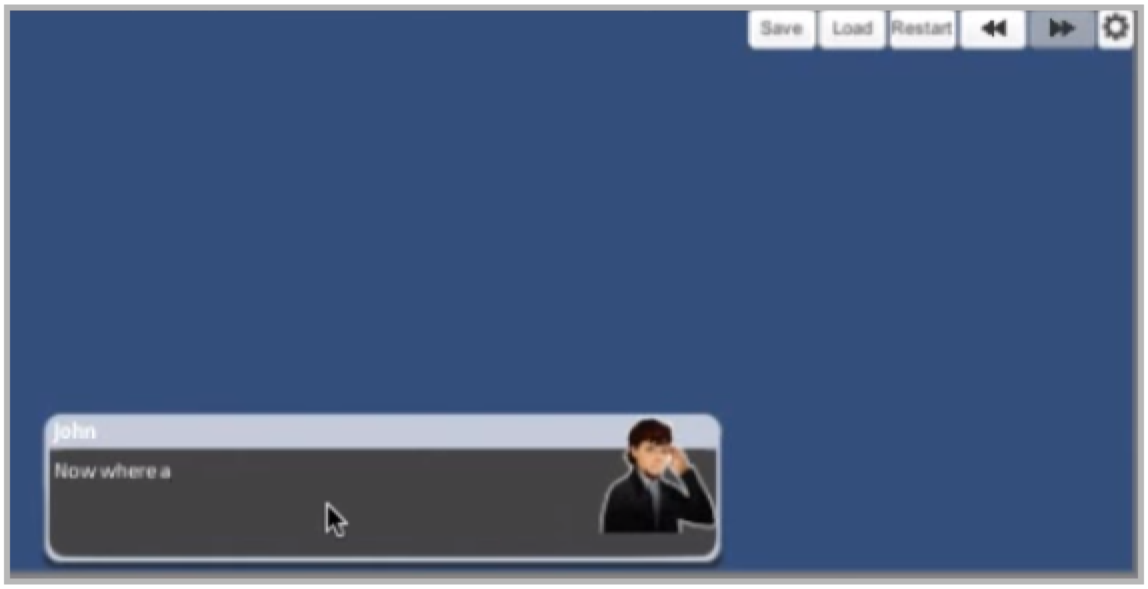


Figure 13: A screenshot of Fungus' built-in save system which can be seen located on the top right of the image. Users can save, load and restart their game through this menu.

Solution:
Fortunately, CG-Tespy has graciously released a slot-based save system on Github to replace the built-in Fungus save system. This save system is closer to other more typical save systems and has gained lots of popularity within the Fungus community. Check out the Getting Started section of their wiki to read about how this save system works and how to implement it into your game. There is also API documentation available on the specific commands and methods included in this package as well which can be helpful.

Resource(s) Used:

- https://muut.com/i/fungus/fungus-development:created-a-free-slot-based-s
- https://github.com/CG-Tespy/Fungus-Slot-based-Save-System

Case 5: Quick Time Events

A **quick time event (QTE)** is when the player needs to complete some interaction, such as pressing a specific button, within a predefined time in order to trigger a certain event. Usually QTEs result in multiple paths (ie., if you complete the QTE successfully vs. not, there are different outcomes). QTEs are a possible way to add interactivity and actively engage the player. Note, there is a time and place for QTEs since they often induce feelings of stress and add pressure as they test the player's reaction time and quick decision making skills.

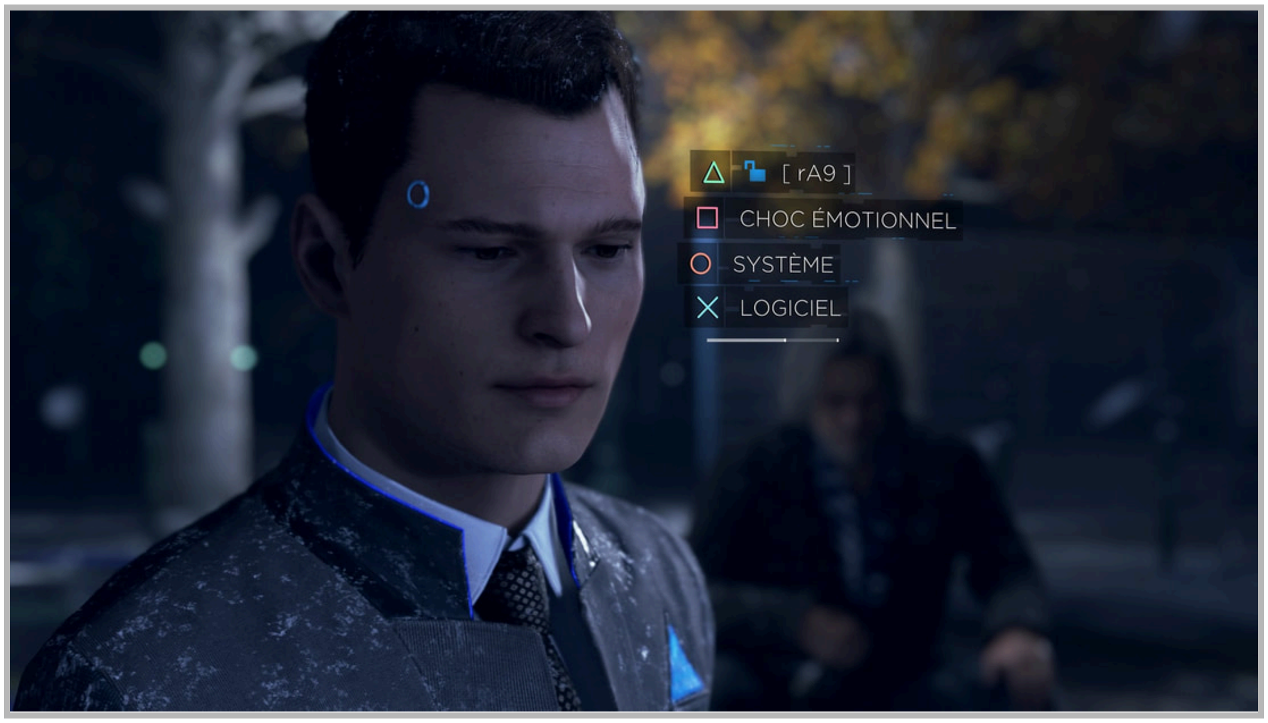


Figure 14: A quick time event from the adventure role-playing game [Detroit: Become Human](#) (2018) by [Quantic Dream](#). The time to complete the QTE is visualized by the decreasing white bar below the four choices available.

Problem:
There is no built-in QTE functionality within the Fungus plug-in. Additionally, none of the in-box commands within Fungus seemed to be relatively easily adapted into creating a QTE. There are QTE systems within the Unity Asset Store and some discussion in forums on this topic but nothing about a QTE system that works with Fungus and its base components. With our current skills in C# and scripting in Unity, building a QTE script or adapting an existing script to work with Fungus seemed like a daunting task with questionable feasibility.

Solution:
Luckily, after lots of searching, we came across a custom Fungus QTE command script made by Albert Gao within a public Git repository. This script includes the components needed to create a QTE scen. These components include a count timer (how long should the QTE last), QTE button (the button the player needs to press within the time limit), block for when the QTE is failed as well as for when the QTE is successful. Although we have yet to implement QTE's within our own prototype at the time of writing this document, we intend on using this script to recreate a rather stressful decision our patient character needs to make when they first arrive at the ER within our game.

Resource(s) Used:

- https://github.com/Albert-Gao/Fungus-QTE-Command

Conclusion

First of all, congratulations on making it through this document! We understand that we covered a lot here but hope we were able to provide you with a basic understanding of what you can do with Unity and Fungus (in terms of creating narrative games), where/how to get started, and helpful troubleshooting tips.

If you have any questions about anything you've read here or would like to connect, feel free to reach out to us:

Livia Nguyen - livianguyenvisuals@gmail.com

Linda Ding - nanlinda.ding@gmail.com

Good luck out there! 🍀

Resources

This is a compiled list of some additional resources we used throughout our Unity, Fungus, and narrative game creation learning journey. Feel free to check them out!

Online Courses

Freeman, J. (2016, December 7). *2D Game Design and Development Essential Training*. LinkedIn Learning. https://www.linkedin.com/learning/2d-game-design-and-development-essential-training/welcome?u=56982905

- Time: 1hr 9min

- Overview: Includes picking an engine, building artwork, and incorporating sound, to publishing and marketing the finished game.

Freeman, J. (2016, December 19). *C# for Unity Game Development*. LinkedIn Learning. https://www.linkedin.com/learning/c-sharp-for-unity-game-development

- Time: 1hr 37min
- Overview: To learn the structure, syntax, and language of C# as it works inside the Unity IDE along with variables, methods, data structures, flow control, classes, inheritance, interfaces, and composition.

Unity Technologies. *Unity Essentials*. Unity Learn. https://learn.unity.com/pathway/unity-essentials

- Time: 11hr 15min
- Overview: Introduction to Unity, it's capabilities and how to get started.

Unity Technologies. *Junior Programmer*. Unity Learn. https://learn.unity.com/pathway/junior-programmer

- Time: 38hr 40min
- Overview: Learn about fundamental programming concepts such as variables, functions and basic logic through two practical projects.

McGrath, P., & Gregan, C. (2017, August). *Make Unity 3D interactive games with Fungus - no coding!*. Udemy. https://www.udemy.com/course/make-interactive-games-with-fungus-unity3d-no-coding-required/

- Time: 7hr 57min
- Overview: Walkthrough on how to build narrative story games, point and click adventures, and hidden object games using Unity 3D and Fungus.

Unity Technologies. *Beginning 2D Game Development*. Unity Learn. https://learn.unity.com/course/beginning-2d-game-development

- Time: 17hr 35min
- Overview: Learn how to build a 2D game starting from a 2D Game Kit as well as from scratch.

Unity Technologies. *Creative Core*. Unity Learn. https://learn.unity.com/pathway/creative-core

- Time: 19hr 45min (Missions 6-10)
- Overview: Learn about cameras, post-processing, audio, UI and prototyping in Unity (Missions 6-10).

Readings

Heussner, T. (2015). *The game narrative toolbox* (1st edition). Focal Press. https://doi.org/10.4324/9781315766836

MCV. BizMedia. (2013, November 4). *Designing game narrative: How to create a great story*. MCV/DEVELOP. https://www.mcvuk.com/development-news/designing-game-narrative-how-to-create-a-great-story/

Skolnick, E. (2014). *Video game storytelling : what every developer needs to know about narrative techniques* (1st edition.). Watson-Guptill Publications.

Videos

- LinkedIn Learning - Unity 5: Build a Character Dialogue System
- Spiritfarer Documentary - A Game About Dying:
  Spiritfarer Documentary - A Game About Dying
- Game Developers Conference (GDC) Talks:
  - Writing and Narrative Design: A Relationship
  - Storytelling with Verbs: Integrating Gameplay and Narrative
  - All Choice No Consequence: Efficiently Branching Narrative
  - Press Y to Cry: Generating Emotions in Videogame Narrative
  - Choice Architecture, Player Expression, and Narrative Design in Fallout: Ne...
  - Stories that Haunt and Heal: Mental Health and Game Narrative
  - Storytelling Tools to Boost Your Indie Game's Narrative and Gameplay
  - Evolving Emotional Storytelling in thatgamecompany's Sky
  - Life is Strange: Music in a Narrative Driven Game

Additional Optional Resources

Carnes, B. (2021, April 15). *Game Development for Total Beginners - Free Unity Course*. freeCodeCamp. https://www.freecodecamp.org/news/game-development-for-beginners-unity-course/

❖ Covers how to install Unity, an overview of Unity, the basics of C#, and how to create a complete game from start-to-finish.

Halliwell, S. (2019, October 24). *Tutorial Videos*. GitHub. https://github.com/snozbot/fungus/wiki/tutorial_videos

- List of links to various Fungus tutorial videos.

Linares-Pellicer, J. *Introduction to video game development with Unity*. edX. https://www.edx.org/course/introduction-to-video-game-development-with-unity

- Introductory course on how to develop multiplatform videogames using Unity.

Vázquez, J. D. *Introduction to Unity for 2D Video Games*. Domestika. https://www.domestika.org/en/courses/716-introduction-to-unity-for-2d-video-games

---

Appendix

Games Engine Comparison

Check out our Game Engine Comparison document for an in-depth comparison of 10 game engines we considered for our MRP.

Games To Play

- What Remains of Edith Finch
- Life is Strange (Chapter 1 - free)
- Spiritfarer
- So May It Be
- When the Past was Around
- What Never Was (free)
- Papers, Please
- GRIS
- Detroit Become Human
- To The Moon
- Doki Doki Literature Club (free)
- Heavy Rain
- Beyond Two Souls
- Samsara Room (free)