Designing a Narrative 2D Unity Game with Ink: A Streetcar Named Desire Adaption

Daniel Nguyen

[dknguye5@uno.edu](mailto:dknguye5@uno.edu)

1. Introduction

Unity is a game development program that has been used to create many games. Among them are games with many player interactions with other non-player characters (NPCs) and objects which can be described as narrative games. These interactions can take the form of a dialogue panel or some other boolean or value as a result of that encounter.

The text editor Inky offers a Unity compatible system that allows for a more streamlined choice-based system and metadata to be hidden behind dialogue. The original system used default .txt assets that ended each current line with a /n character. Inky has a much more intuitive system with support for choices and tags that is easily accessible through Inky’s custom Story datatype.

My aim for this project is to create an interesting world where the player can explore and interact with most of its objects.

1. Motivation

In choosing a topic for my game, I decided on using the Tennessee Williams’ play “A Streetcar Named Desire”. The play is about a woman named Blanche who moves into her sister’s home in downtown New Orleans. At first, she mostly gets along with her brother-in-law, Stanley, but through her many insults, he is provoked into searching into her dark past, which ultimately causes her untimely downfall.

Streetcar offered an interesting setting, with interesting characters and already included many great lines that made it into this game. I planned on building on many of its themes with more paths the player can choose from as he/she plays as the main character, however only managed to mostly recreate one of its scenes.

The game is based on most of Scene Three, cutting out the end. Most of the lines are copied from the source material, but I have added a few alternate options, although these are mostly unfinished.

I hope this game acts as a faithful adaptation of Streetcar and also hope to complete it one day.

1. System Description

The main component of this game will be the Dialogue Panel (and every child object under it) and the scripts used to activate and update the panel. Starting with the Dialogue Panel, it is designed with Unity UI objects like Panel and Button. Included under the dialogue panel is an animator for any portrait changes, a TextMeshPro textbox to display text, and buttons to function as choices when needed by the dialogue. Other than the Dialogue Panel, this narrative game will have the usual aspects of other Unity games, in this case 2D Colliders, 2D Rigidbodies, and sprites.

All these basic game objects are placed under the Grid folder and can be grouped into three categories: Background – Floor, Background – Furniture, and Characters. The Background – Floor tilemap acts as a layer for all the tiles used by the Unity tile system and has objects acting as colliders for the walls as seen in the game. While the Background – Furniture tilemap has more solid objects with built-in colliders like the bed or chair. The Characters folder has all the objects that would be considered “characters”, each with its own collider and rigidbody. Also note that I have the transparency sort axis as based on the y-axis, which explains why some objects appear closer or overlay other objects when having a higher or lower position on the y-axis. For example, the character objects of Player and stanley are on the same sort layer, but when the player is closer to the camera or has moved forward, he/she will be rendered on top of stanley and vice versa.

I will be using the package Cinemachine’s 2D virtual camera with the default main camera to utilize more camera functions. For example, the main camera does not come with an functionality for following an object, but the Cinemachine camera allows me to set the camera to follow a certain object. The virtual camera is also modified to frame an average size of 1 and have all the other default settings.

Any audio and timeline assets in use by the game are under their respective managers in the Managers folder or in other relevant folders. A manager is defined in this game as an object with an encompassing script that manages any objects associated with this script in an appropriate manner. For example, the audio manager has a script that takes an audio file inserted by another script and plays it. Although the game does not currently have any background music, there is also a background audio manager that could play another audio file while the main audio file is playing.

The other manager is the dialogue manager and much more important. Like the audio manager, the dialogue manager has a script of the same name that uses many other preassigned game objects to determine what data needs to be assigned when and where on the Dialogue Panel. For example, the dialogue manager is loaded with public variables for the Dialogue Panel and its buttons, to be able to assign new attributes to those variables as needed by the current text.

Since Unity does not draw its assets from the Project folder, any assets needed will have to be found within the game itself, and since those assets can’t exist on their own, these assets are assigned otherwise empty game objects and can be found under the Audio Library and Timelines folder.

The final assets to note in the game are found under the Audio Library and Timelines folder. More on this in Methodology, but these folders store all the sound files associated with voice lines and all the timeline files associated with movement in the game. Also note that any text assets are stored externally to the game, but are assigned to scripts to activate them.

The game’s functions mainly revolve around detecting some player interaction and executing an appropriate action based on the assigned text. Here is the flowchart of the game:

Diagram

Description automatically generated

Starting with detect player input, the game will wait for the player to make a move toward an interactable object. In this example, the radio object will contain a Dialog Activator Script with the text object as “mitch” and Booleans “Require Button Press” and “Destroy when Active”. The text object is the .json file that is the compiled result from an ink file. While the “Require Button Press” Boolean requires the player to press a button, in this case “e”, to trigger the dialogue and the ”Destroy when Active” Boolean deletes the script and object containing the script to prevent the player from activating the same dialogue again. When the player moves a move toward this radio object, it will trigger the OnTriggerEnter2D method that will detect the player’s Rigidbody2D and Collider2D and activate that script. Since “Require Button Press” is active, the game will require “e” to be pressed to activate the dialogue from another script and upon pressing “e”, the “Destroy when Active” Boolean will perform its tasks.

Once the Dialog Activator Script has been successfully activated, a method call will be made to the Dialogue Manager Script with the preassigned text object. The first method to be run in Dialogue Manager is EnterDialogueMode(). This method sets the Dialogue Panel to active and assigns the current line as a new data type from Inky called Story. The Story datatype makes use of a number of methods similar to that of an iterator in a singly linked array like canContinue() and continue(). Using these methods, the script assigns the text on the Dialogue Panel to the current line in the Inky file and moves on to displaying choices.

Text, letter

Description automatically generated

The DisplayChoices() method first determines whether the Inky file has any choices and if so, assigns those choices to a list and, from there, the appropriate buttons in the Dialogue Panel. Then it waits for the player to make a choice. Here is a button from Dialogue Panel, upon pressing the button the Player will pass a 0 to the MakeChoice method which will call the Inky Story method ChooseChoiceIndex() with 0 to choose the choice set as 0 on that list.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Once the script has properly assigned the current line and choices, it handles any tags. Tags are declared in Inky as any text past a hashtag symbol “#”.



In this example, you can see the voice, portrait, and camera tags. For the line “Who turned that on in there?”, the voice tag will then have the script play the stanley\_16 voice object through the Audio Manager.

Handling tags take in the currentTags List from the Inky Story object and splits each tag around the colon “:” into a key and value. From there the key is sorted into a switch statement with enums. Once a key has been sorted, the script will change whatever value to the value on the tag.

A picture containing text

Description automatically generated

In this example, the voice tag is detected and the assigned audio manager is called to change the music to an audio source file of an object with a name matching the value. In other words, if the tag is calling for “stanley\_16”, the script will search the game for a game object matching that name and call ChangeMusic() with it.

Once the current line, choices, and tags are handled, the player is handed back control where the player has to press “Enter” or “Return” to continue or end the dialogue.

1. Methodology

The programs used in this game are Unity, the text editor Inky, and MS Virtual Studio for editing scripts. As explained in System Description, Inky has the Story datatype and Lists for currentChoices and currentTags that make organizing information from text very straightforward.

I started this project only knowing Java, but C# came naturally for me when writing the scripts, and I used a lot of YouTube videos and Google to answer my questions and improve the game. There was this excellent series of YouTube videos on Inky by the channel Trever Mock that first exposed me to Inky and showed me some of its many additions.

1. Results

Since I do not want to potentially infringe on any Unity rules, I will ask you to see the class presentation for more information.

1. Conclusion

Although not finished in this state, I hope you can see the promise it holds for a truly engaging experience, with varied tags adding to or triggering different choices that will impact a player’s final version of the game.

1. References

As references, I would like to refer you to the full catalogue of games designed by Inkle [1], but here will dive more specifically into the game 80 days.

Graphical user interface, website

Description automatically generated

This is a typical scene in that game where the player is traveling with his boss toward what appears to be Aden, Yemen. In this game, you are a servant managing your boss’ trip around the world and must decide based on several factors like available routes at certain times, money, and more. On the bottom left, you can see a button to open a menu for the inventory and on the bottom right, you can also see your boss’ health which you need to maintain. Also consider the limited time per day you can travel and rising costs for luxury travel. While this is going on, you also have to manage your conversations with the locals. In this example, the player depicted on the left is talking with a pilot about routes in the local area. He must determine which route would benefit his mission best, i.e. whether his inventory or boss is best suited in that region and also how much time and money it would consume.

In designing this game, I based my thought process around the idea of managing multiple factors through dialogue on 80 days, and tried to manifest this through Inky’s tag system.

[1] <https://www.inklestudios.com/>