They Were Doomed from The Beginning Post-Mortem

Table of Contents

Introduction

Doomed from the Beginning, is an Interactive story with RPG elements, with an ever-changing story mechanic and narrative combat. The story starts with the player drawing a card from a Tarot Deck, determining their character, they can draw additional cards to give the player alternative weapons, magic and relics that affect their stats and the story from the view of the character they've chosen.

The idea itself required work to design, the two designers who had pitched the idea to me had clearly done the legwork, they had a rough idea for what they wanted, the rough length of the story, and the overall content and features they thought the game required. These changed a bit early into preproduction as scoping was a hurdle for this product, with only one dedicated programmer and one dedicated artist, our work was cut out for us.

This project was new to me, and not something I had experience with, my experience with Strings and Regex in C# were both limited, only using them for name storing, saving data. This required me to research solutions to pre-process strings, create a system that could write out a story the same way every time, while also being flexible that it could manipulate these inputs to provide additional context, to expand from the story, and to provide first-person view from three characters. The systematic need for this game was very large. We needed to tell a story, that was repeatable, while also changing and varying in its outputs.

Programming

How do we tell the same story, differently?

We can do this a few ways, but many would restrict the kind of content we add to the story, and complicate changes later in development. The means that I went forward with was using XML Table sheets, XML has many problems, first off, it is not easily readable to regular people, if you've worked with it in the past, you quickly pick up what you need, but other formats would be better for creating, such as JSON which, would be much more readable, similar performance and easier attributes.

Had I known of JSON, I would have utilized that option as the improved readability would have solved several issues the project had when importing information, and the additional formatting characters that XML tends to throw in. Using XML over JSON or other formats, didn't hurt the project, but it could have prevented a few problems, and possibly helped the setting up of importing.

Inserting personalized elements

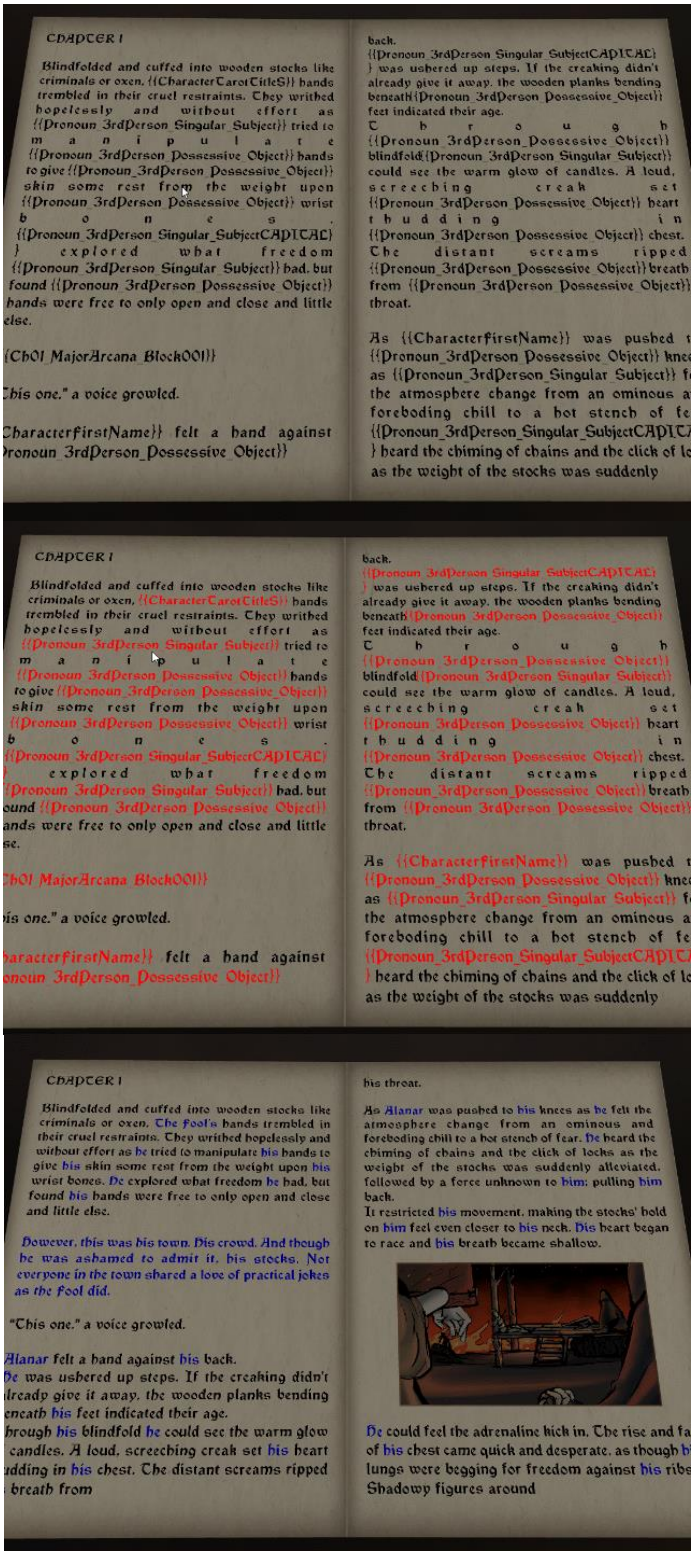
Some time went into thinking about how we went about this, and I'll try to go over what I've done to achieve my results, they aren't perfect, but at least worth explaining, I'll go over the first page of content as "The Fool" and explain the steps.

1. The story is built with an overarching theme, with characters weaving themselves in and out of it, without losing the message.

Each character is imported through XML with many tags, using dictionaries to processes these tags. To find each tag, we use Regex (Regular Expressions), originally, we were searching every word in the story, while computers have the power to do this, it is greedy, and open to be broken if inputs aren't setup in the correct way.

2. As we use regex, we can search large portions of text for anything within {{ and }} quite inexpensively, we search the inner text in the Dictionaries mentioned earlier, when nothing exists, it sets the colour to red, which assists with debugging, if it is found, we concatenate before {{ with the replacing values, and continue the text after the }}. This works efficiently, and with a moderate density of tags, I can process upwards of 3,000 characters without delay. We only process one Page per frame to avoid long stalls. (Including other processing required)

3. In the picture shown to the right, you can see the result, highlighted blue to show text replaced to fit into the story, Character name, how the character expresses themselves, him, her, foreshadowing thoughts. This can be further expanded through Ticker Queries to great effect.



With only one programmer, we had to shift the application work that would be involved with creating this story to the designers, they knew the story, and understood the lore better, the three of them had the resources, to allocate themselves to work on XML, while the others could work on the main story elements. The choice was a success, although overall, it created additional work not foreseen, it allowed me to focus on other areas, and stream line more important processes like page breaks and interactivity of the scene, than if I had chosen to import the work myself.

Tracking History & Querying History

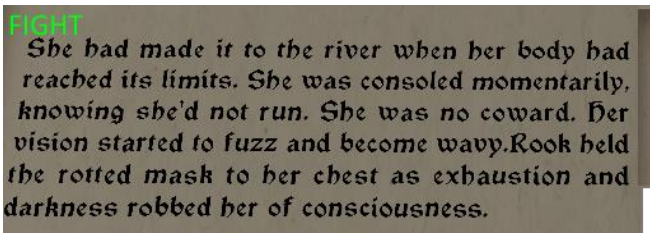
The introduction of Inline Queries, how do we know the player has been hurt > 4 times? Does the player have a companion? Did the player flee an event they could have easily won? is the player holding a certain weapon? Certain gender? The list goes on, Inline Queries, gave the game a flexibility in late-production, it allowed a core story to be developed, then queries could be created during testing to give the story additional depth and meaning.

We created these Queries through XML that use a small class to manage the inputs into a desired output. An example can be found below

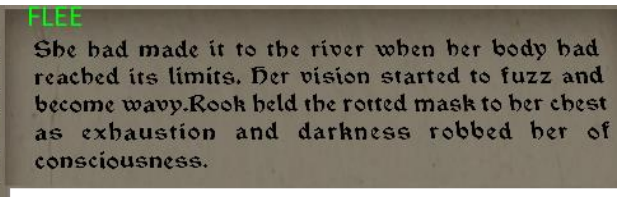
```
<QUERY ID="Ch01_Ticker_001">
  <SUBQUERY>
    <ID KEY="Ch01_Event01_ChoiceA" TYPE="=">1</ID>
    <PASS> {{Pronoun_3rdPerson_Singular_SubjectCAPITAL}} was consoled momentarily, knowing
    {{Pronoun_3rdPerson_Singular_SubjectD}} not run. {{Pronoun_3rdPerson_Singular_SubjectCAPITAL}}
    was no coward.</PASS>
  </SUBQUERY>
</QUERY>
```

During the first event, you can fight, or you can flee, if you fight, even if you lose, a portion of text will be added to the story, this could be expanded further for each character, or even for the weapons they have.

FIGHT



FLEE



This is all done inline, with queries made against the Ticker, that events and story elements interact with, adding or removing values from the ticker as the story progresses. The Ticker itself is quite simple in its principle, if we want to remember something that happened, we give an Event/Story/Action a memorable Key that is used and stored, with a value and is done to allow freedom of expanding the story, it gives designers the freedom to bring up a small thing from early in the story, to the present, if playing as the fool, and your dog companion died earlier in the game, the character will reflect on that due to the Ticker values expressed.

Characters, items & even the Queries can add values to the Ticker, this can also create additional problems through storytelling as it can bring a reliance on storytelling to be built later. As it may become more desirable to create a bland base story, then filling the story with the content through tickers. I don't believe we ran into too much, but ticker presence in the story in the final chapter is significantly higher, but can also be due to the more ticker values that can be referenced. Overall, we didn't add nearly as many ticker queries as desired, which, could flesh out the story further, but we feel enough has been added that it shows off what is capable.

The Ticker system itself, was a great addition to the project, although Story Elements and Events can alter the path of the story, had we moved forward with the choice to branch our story outward trying to mash it back together at the end, it would have been a mess for the designers trying to bring it back and many elements would be repeated to fill additional choices, proofreading would be required to avoid doubling up. The Tickers ability to store and handle simple questions allowed a main story to be made (Branches could still be adapted when desired), but the filler, and personalized content could be driven while proofreading. The system could be improved by having an idea of its implementation before proofreading, having an idea for what it is we want to be driven by the ticker and its queries, however, time was a limiting factor.

Showing Cause & Effect

We can show-off a few things in a story, we can play audio inline, or we can Shake the cards in play. During story elements, when we want to shake the players card a bit, to indicate that something is happening 'in the moment', and I'll try go over how it works.

Shake Shake Shake

Providing a Key for the Dictionary that would fire the processes, we also use a few attributes in XML to store additional values, a screen shot of our Shake Minor process is below.

```
<SHAKE ID="ACTIVATEMINOR" KEY="MINOR" INT="5.0" TIME="0.5"></SHAKE>
```

ID is the key of the Dictionary, KEY is the Key to the Card we want to shake, we have predefined values for MINOR & MAJOR which will shake all Minor/Major arcana cards, INT is the intensity and TIME is the length that the card shakes.

This was the easiest part of the processes, as to fire these I need to calculate where the text has been writing to, and find an Invisible tag that we can't show the player. The end result involved counting the visible characters each frame, and shaking the objects when we were ten characters away, or past the value, this involved storing the times elements had been fired to avoid accidentally triggering processes multiple times.

It looks good, but was poorly implemented on the designer side, and poorly explained by my side, overall, although it is visible and can be shown off, most people aren't going to understand what it is the shaking represents, with more time, I feel like a hint text message should have been used to indicate to the user what a shaking card can indicate.

Text Formatting

The project used the asset TextMeshPro, to make this project possible in the time we had, but it also gave us RichType text tags, the story can use special tags that would create an end line, or "<color=red>Red Text</color>" would result in Red Text. Simplifying the process of formatting text, story elements can reference certain tags stored in XML and replace chunks of the story, and each chunk would then be processed as well. This led to a branching story, but intentionally those branches fall back to the base to prevent complicated scenarios where the final logic is impossible to predict.

Allowing actions that are a direct cause of an item, to light some text, or italic a Quote, are small things, that together, really improve the readability and the Effect of action in a story. The use of TextMeshPro left me with time to work on other tools, as it provided most of the foundation to get a good-looking story working.

Project Successes & Failures

Failures

Using TextMeshPro without enough research

While researching during pre-production for possible solutions to display large amounts of customized text to a user, I discovered TextMeshPro, a flexible and free asset that can show much more text at a time than Unity's default Text Meshes, TextMeshPro is clearer in display, and has additional features, such as a page overflow which sold me on the asset right away. The game contains six main Meshes for displaying text to the player, with usually four or less active at once, two when pages are not turning. I need to be able to update page contents and flick the page without noticeable drops in performance. The problem here is, I didn't account for the vertices, although looked great, each mesh was storing all the text, and generating their related data and showing the text, however, it was also generating information I didn't need, such as Character & Page information, and other things we do not need at any immediate moment. Any frame there was a change to text, which, was about every second frame at default typing speed, this led to noticeable performance loss when text meshes were being turned. On three pages full of text, could result in tens of thousands of vertices quickly chugging the game to a crawl when moving, even when only 5,000 were visible as their positional data is marked dirty due to the needed rotations, and iterating over thousands of characters of text.

By now, we were 5 weeks into Production, moving towards Alpha, and to fix these issues, I created two Text Meshes, one for the story, one for processing. Both are kept out of camera view, and static to ensure vertices could not be dirtied by object movement. The main class in charge of managing this had to be rewritten to work using the new method of working with one core mesh. While also allowing us to cut it up when it detected overflow, catch any RichType tags and ensure they were not broken in the middle of tags, also bringing colour across when detectable.

Resulting in the creation of a Page Content class which stored Raw Text, Processed Text & Character counts, which allowed me to type pages better, while providing me the ability to Slice and Dice pages as needed, the result is still not great, with some pages cutting off one or two words onto the next page before ending the paragraph. With more time, this could be solved, but I had already spent another week by this point developing a new technique of processing text. The result, was text based vertices never lifted past 30 thousand, and performance was factors of magnitudes better.

Team Size & Sickness

A team of five, was simply not large enough for the project, more out of our control than I'd have liked it to be, having an additional programmer could have solved many problems we ran into while testing, having to create a process to break pages into chunks and display those chunks in repeatable ways while also saving, storing, and reprocessing when required. Having little experience in String processes and Regex commands, time was spent researching how to go about the more efficient routes, that wouldn't upset the Garbage Collector. During these moments of research, the active programmers dropped to zero, when new tools and features were not being created, due to a limitation of knowledge.

During the production before alpha, a few team members were ill, which dropped our available resources significantly, with us having no artist for a week and a half. Fortunately, we did fear this becoming a problem, and had scoped for a smaller project than we wanted, and the overall organisation of the team, quite probably kept the team afloat during the week where only a couple of us could work at full speed.

Lacking Forethought

The game was unique from the start, and many early questions were "What makes it a game?", and although the designers had their answer, what made it a game was not thought through until later than what would have been ideal. They knew what they liked about this kind of game, but didn't think too heavily on what it was others would enjoy about the game. The game itself relied on the user knowing storybooks, with choose your adventure, although not crippling, this could lead users not to understand what it is the game is doing as they progress through the game, and it isn't clear until well into the second chapter that your choices do anything unique.

Furthermore, the more gamey aspects weren't completely thought out until before Alpha, the events were repeatedly adjusted throughout development due to new thoughts and mechanics, although not overbearing, the choice to make events based more on random values and iterate over a dice roll approach required some ugly changes in code that were not cleaned up late in development although still before Alpha, there were other features that still had to be developed.

Originally, the events had you choose your action, and it checks your ticker like usual, "does the player have at least 4 might", when yes, you pass, when no, you fail. The new approach, you check with multiple tickers, do average the stats, if it is higher, they have a better chance to pass, if equal, a bit lower chance, and if less, they have a very low chance. This resulted in changes to the XML to allow for adjustments to the random scores given to these values, and further complicated the event container, it was now storing three pairs of two strings for pass/fail options during these dice rolls. The result looks great, and feels more like an old classic game of you attack, they attack, it is a mechanic that should have been birthed early in design, and not after the core of the events functionality had already been created.

Successes

Communication

Being a small team, communication was important, and working in the same room every day, helped us be fruitful greatly. Although our schedules could have been better, our communication at the start of the day allowed each stream to work independently without requiring too much of other streams throughout development during the day. The ability to know what was needed in what order pushed us through the crunch period without much stress at all. However great the communication was, I feel we could have improved upon our scheduling as that will reflect poorly in real business applications.

Sufficient Scoping

Thankfully, the project was scoped down after the interview to get it approved, with ideas and suggestions from others interested in story writing and game production. This gave us a better feel for what it was we were taking on board.

Both Art and Story design were scoped down a fair bit to ensure we could have a minimum viable product at least done by release. We had to scope a bit further down for the story due to illness' but the overall product is still a great peace, with few bugs and issues.

Ticker Queries

These were thought about before the interview to see if we'd get the green light, the concept is simple, we want to store values progressively through the story, and we want to do stuff with these values. The only complicated part was the dynamics of the problem, how do we know what we're storing, and how do we do stuff with that may be null value?

This turned out to be quite easy, fortunately I had decided to implement queries not long after I had a basic text replacement system for Characters and Items, guiding me to flow straight into the XML solution, we give story and event scenarios the choice to store anything they want. This gave designers more room, as they only had to add a new key to events and stories to provide these values, a Key like regular text replacement, and the value. The result was any time something was added to the ticker, the value of xml was converted to integer, then added/subtracted the value if + or -, and worked well, but I should have probably left the ticker to work with just integer once it was loaded in-game, there was no need to store it as a string. The overhead was quite minimal, and overall just something overlooked during development.

The queries itself can do basic manipulation, and be asked simple questions, when it is equal or greater than the value suggested, basic setup through XML and a key to fire the question, with a pass and fail scenario, like events, but inline. It works great, and if setup correct, isn't noticeable that it is happening which gives a greater fidelity to the game.

General Room for Improvement

What we as a team accomplished in the time we had, and the hurdles that slowed development, I feel we accomplished something both unique, and of acceptable quality. If given the chance to start over, I believe we could have improved upon our processes in many areas.

Throughout development, a common wall to progress was understanding about how to import the story and interactivity to the game through XML, more time would have gone into thinking about how it will be used. Some of the Tags and XML processes could be simplified, how events are handled is broken into a core 'event' and 'responses', each possibly changing ticker values, each possibly stating the direction of the story, and how this all worked was not made easier by the way it was all implemented.

More time could have been spent, likely through meetings, where we could have discussed how the XML was working and how the XML sheets interacted with the story, early problems both myself and the designers ran into was formatting, Intuitive storytelling where spacing paragraphs doesn't look or act as you'd expect, you can't just use two formatted end lines and expect it to work. The page is alive, and the next time the game is played, that line may not end there, it may end up on a whole separate page, or not even show up at all. Will the story still need that space? This was most notable during Events, when special event line spacing was used, if a story element fired a spacing format break just after an event, or just before, it would space the event weirdly. Formatting characters were simple to implement in code, but hard to execute in a way that looked right, for me, as the programmer, and the designers, more time should have been spent talking about how certain formatting should be used.

Conclusion

We worked well as a team, without much conflict throughout development, and although the teachers had their worries for the project, their support at the initial pitch greatly helped us in being given this chance to develop Doomed from the Beginning. Had we another chance to redo this project, much could have been improved, however, that could be said for any games created this year.

Time was an asset we overlooked, and team size restricted a lot of the content we wish we had for the final product. Although curious as to what we could have produced with another three artists, and an additional programmer, I'm proud of the product we produced, and have learnt more than I had ever expected from this project.

I wish all those involved with the development of They Were Doomed from the Beginning, and the teachers who gave their input throughout development the best in their future endeavours.