The Disciple: A Talking Platformer

Senior Project submitted to The Division of Science,

Mathematics, and Computing of Bard College

By Ben Sernau

Annandale-on-Hudson, NY

May 2017

For Mom, Dadys, El, Shad, and the Chief

Acknowledgements

I thank Noah Segal-Gould for telling me how computers work, since I am very bad at using them. I thank Keith O'Hara for giving me good assignments, making me finish my work on time, and reading this paper. I thank Sven Anderson for his legions of academic textbooks, without which I would still be confused. I thank Robert Mills, Noah Segal-Gould, Elias Posen, Charles Calder, Dana Lubow, and Derek Low for giving me additional experience in the world of Unity. I thank Eliot Sernau, Aaron Krapf, and Michael Callejo for playing the first two levels of my game and giving up because it was too hard. Finally, I thank Tanner Cohan and Eliot Sernau for offering a soundtrack. While I did not need one ultimately, I appreciate the support.

Table of Contents

Abstract

Working in Unity to create a two-dimensional platformer with a Natural Language Generation system, I have considered a new way in which Artificial Intelligence may affect gameplay. The resulting project, *The Disciple*, takes input from the environment of the game and offers successfully a sentence relevant to what occurs within the game's world. The sentences this system generates are diverse enough so that, while the Natural Language Generation system may restate what it has said, already, it does not utter the same sentence twice in a row. Often, the Natural Language Generation system selects a phrase I have written from a large directory of sentences. Occasionally, it sends information to a syntactic component that builds simple sentences, itself. The syntactic component succeeds in creating different parts of the sentence and putting those parts together in order to complete the sentence.

Introduction

Modern games are technological marvels. These massive, multidisciplinary projects have been offering new opportunities in the realms of narrative and expression since the late 1990s. Gaming has matured from a yellow circle who eats ghosts into thrilling Indiana Jones-like stories of treasure hunts and space-age epics about human survival. A company might have dozens of programmers work on a game, and these programmers are joined typically by writers, musicians, and artists. One need only play one of these high-budget games for several minutes to understand the work that goes into them. Take *Assassin's Creed*, for example. The assassins are running around a medieval city. There are hundreds of people walking past them. Sometimes, a bird flies overhead. The non-player characters are walking, dancing, or talking to each other. These minor details are easy to take for granted, since the C++ files are not visible. What game programmers need to accomplish is admirably difficult. Still, a common complaint from the gaming community with respect to high-budget games is that the stories in these games are usually uninteresting. There exist some games that offer stories appreciated by gamers across the board, such as *Bioshock* and *Halo* (a quick Google search points to outstanding acclaim for these two games), but because a game with a good story is so difficult to find, my senior project seeks to investigate what more a game can offer in terms of narrative.

Before the discussion continues, there are some concepts that are worthy of elucidation. The type of game I have created is a two-dimensional platformer. This means that the main character moves from the beginning of the level (usually on the left side) to the end of the level (usually on the right side). This is because the world of the game exists in two dimensions. One may either progress to the end or regress to the beginning, and one can increase or decrease height, but one cannot move into or out of the screen. Games from the late 1980s and the early 1990s exemplify this concept best. Some of the most famous ones are *Metroid*, *Castlevania*, *Super Mario Brothers*, and *Sonic the Hedgehog*. Such games feature typically sprites, computer graphics that may be moved on-screen and otherwise manipulated as discrete entities.

The second concept to address is that of Natural Language Generation, though more detail will be offered by the following pages. Natural Language Generation is simply a subfield of Artificial Intelligence wherein one makes a computer appear intelligent by equipping that computer with lexicons and grammatical rules. While this type of Artificial Intelligence has not appeared in a two-dimensional platformer, before, it has been in a game. In *Facade*, a game by Michael Mateas and Andrew Stern, the player types sentences into a terminal in order to mediate an argument between two artificial parties. The game ends when both of these artificial parties reach either an agreement or an agreement to disagree. The player communicates to these parties and persuades them by typing sentences into the terminal. In this game, the Artificial Intelligence is smart enough to offer incredible diversity, and feeling like one is talking to actual people is not

difficult to do. As an undergraduate, I have attempted to bring Natural Language Generation to the realm of two-dimensional platformers at a smaller scale.

The purpose of this project is to explore Natural Language Generation within the context of a game. This paper is a discussion of the feasibility of a Natural Language Generation system that works with a two-dimensional platformer in order to generate text on the screen. What follows is an evaluation of what my Natural Language Generation system can and cannot do. The discussion includes references to previous projects and experiments conducted by scientists. It addresses the ways in which the current project has built upon previous projects, as the project takes inspiration from many sources. The primary data are screenshots of gameplay, many of which include the Natural Language Generation system's output. This discussion concludes with a consideration of future work.

Related Work

Natural Language Generation and Processing (NLG/NLP) is a division of Artificial Intelligence wherein a computer attempts to communicate with a user in that user's language. With rules for both syntax and semantics, a computer may at least mimic human communication. First, the computer considers the meaning, the semantics, of what it wants to say, then it converts that meaning into words, linking ultimately semantics to syntax. For example, when something happens in-game while someone plays through my project, the computer takes note of that event by storing information in variables. These variables store either Booleans (true/false values) or decimal numbers. With this input from the game's environment, the computer chooses whatever words and whatever parts of speech it needs in order to speak to the player.

This journey from semantics to syntax consists of six parts, according to Professors Ehud Reiter and Robert Dale in *Building Applied Natural Language Generation Systems*: Content Determination, Discourse Planning, Sentence Aggregation, Lexicalization, Referring Expression Generation, and Linguistic Realization. Content Determination is simply a program's decision regarding what information a verbal statement must convey. Discourse Planning is a construction of the order in which a sentence presents information. Sentence Aggregation is the way the sentence pieces information together. Lexicalization is the determination of what pieces of semantics

connect directly to what specific words in any lexicon the program accesses; it is also a consideration of synonyms. Referring Expression Generation is a collection of grammatical designations. At this stage, the computer decides which words are which parts of speech; which words are subjects or objects. Finally, during the stage of Linguistic Realization, the computer finalizes the sentence with any necessary punctuation, grammar becomes applicable to the sentence, and the sentence becomes deliverable to the user. The six stages of this process might seem vague, and that is because they are. What makes NLG so difficult for programmers is that understanding what one must do to implement it is nearly impossible, though other sources shed some light on each of these "steps." First, reducing language systems to templates is not completely unheard of. Consider Reiter's train example wherein templates display a schedule.

```
System.out.println("The next train, which leaves at " +
getTrainTime() + ", is the " + getTrainName());
```

According to this fake Java code, there may exist a template such that there are two pieces of information within the sentence. Otherwise, the sentence stays the same, each time. This method circumvents several of Reiter's and Dale's steps in order to create a system that is boring, but potentially informative. With templates, there is no need to worry about the system producing something that fails to make any sense.

Template-based NLG does not have the variation of human speech, but as far as one is concerned with transforming information into something communicable, it suffices.

Another way in which scientists have attempted to make NLG more feasible is to focus on automating steps. In *Learning to Order Facts for Discourse Planning in Natural Language Generation*, Aggeliki Dimitromanolaki and Ion Androutsopoulos discuss work on a machine learning approach to Discourse Planning, the second step in Reiter's and Dale's process. Recall that this step is about ordering information. Dimitromanolaki and Androutsopoulos built a system that attempts to order pieces of information in the best way during discussions of items in a museum. Museum curators examined the small paragraphs created by the system, switching the positions of sentences if the positions failed to make sense. In any case, Discourse Planning requires human thought rather than the sole computational power inherent to a computer. The rules need to be given to the computer by the programmer, and, even then, the programmer might need to amend the final product in order to have the final product make sense.

With respect to Sentence Aggregation, the paper, *Efficient Algorithm for Context Sensitive Aggregation in Natural Language Generation*, claims that the essence of the step is first to create a set of data for which each datum may be expressed by a single sentence. Then, combining the theoretical sentences, one creates a larger sentence wherein multiple data are expressed by an NLG program.

What is also helpful is to equip an NLG system with informative variables and questions so that managing semantics is easier for it. An "informative" variable is one

that addresses change. For example, as I designed *The Disciple*, I chose variables that would change often. If there is a lot of change, then there is a lot to say, so I placed into my NLG script Booleans and decimal numbers that would either change or trigger often. My variables include a Boolean, `hasFallen`, that determines simply whether the player has been killed by a fall. It switches to true when the player falls, and, otherwise, it is false. When I refer to a "question," I refer to veritable questions asked by the variables, themselves. One may think of variables as "questions" in the sense that each of these variables in the context of NLG poses a question to the game's world. If one is to consider my `hasFallen` variable, this variable asks, "Has the player fallen?" The importance of considering variables as questions manifests itself in the project described by the paper, *Acquiring Correct Knowledge for Natural Language Generation*. In it, Reiter, Sripada, and Robertson discuss a text generator, which creates specifically letters that encourage people to stop smoking. The way the NLG system creates these letters is by having the smoker fill out a questionnaire, then the NLG system is able to personalize the information in the letter by consulting the questionnaire. An NLG system that is able to document many different types of information appears to be more realistic and more thoughtful. The letters the NLG system generates resemble strongly something a human being would write, and they actually offer cogent arguments-- free from logical fallacies-- that incentivize quitting. For the NLG system to offer good arguments, it needs to ask a lot of questions, and it needs to ask questions as specific as, "How soon after you wake up do you smoke your first cigarette?" The greater the quantity and

quality of input to the system, the greater the quantity and quality of output from the system.

There exists an issue of which to be wary, according to the paper entitled, *Emotionally-driven Natural Language Generation for Personality-rich Characters in Interactive Games*. Anything relevant to language cannot be completely undefined. While there need not be a bank of phrases, there might have to be a bank of words (i.e, the entire OED), and the program will need access to some kind of rule set for the sake of grammar. Whether the computer understands what it says or not, the "how" begins with this endeavor. A lot of the time, character believability is a matter of narrative mastery. Creating a believable character is a writer's job. Someone has to pull the strings directly in order to make something happen, and there is no such thing as bestowing autonomy upon the story. All possibilities and cases must be considered by a human being creating the program.

Finally, what is otherwise worthy of consideration with respect to NLG is the division of Content Determination into a qualitative overview of a dataset and a combination of both the qualitative overview and the actual data (proposed by *A Two-stage Model for Content Determination*). The main idea behind this division is that the computer may not be responsible for whatever is open to interpretation; interpretation is the programmer's job. The programmer deals with interpretation by creating paths and rules in the second and third stages of NLG. Anything that one may reduce to both exact numbers and exact words (under any circumstances) is *information*, what one must feed to an NLG program. Another outstanding part of the article is the value of time-series

data, a set of data wherein *something* has some sort of relationship with *time*. Keeping track of the times at which something is true or false can be especially helpful to a programmer seeking to create an NLG system.

In all cases examined by each of these articles, the point is that there is a definite path from meaning to syntax; that language comes from meaning, not meaning from language. When one decides what one wants to say, the meaning behind the words comes first. If one receives information through language, one has to convert the sentence back into meaning. An English phrase is useless to people who do not understand English. Without a pre-existing connection between meaning and syntax, a language is not usable. While not all NLG systems employ Reiter's and Dale's rules in the same order, what remains constant is the propagation of meaning followed by grammatical designations and actual sentences.

Game Design

Unity is a software with which to create games or animations. Because it is easy to use, both programmers who want to cover a lot of ground by themselves and beginning programmers flock to it. As difficult as creating a game by oneself is, Unity allowed for me to cover much ground in terms of what a game is supposed to have (menus, visuals, sound, controls, and physics). The user interface contains everything from a view of the current scene in the game or animation to a sprite editor, and the sprite editor is actually the utility with which I began my project (a significant part of work is the construction of sprites, in fact). Using Unity, one may instantiate additionally entities called Game Objects, which I will address in the following pages. Essentially, Game Objects are the most general constructs in Unity whose flexibility allows for the creation and propagation of many different types of items or characters. Other important components in Unity are Colliders and Rigid Bodies, both of which allow for physics to exist in the game's world. They will also be explained by what follows. Otherwise, in Unity, I could create a script for player movement (among many other scripts), I could create animations, and I could make menus. For reference, the following table describes briefly the purpose of each script in the project. I include the Natural Language Generation scripts, as well.

AudioMaster.cs	This is for general sound control.
ConstantAspectRatio.cs	This is for maintaining a constant aspect ratio.
Enemy.cs	This is for controlling enemies.
GameMaster.cs	This is for control of the menu and saving the game.
LobbyController.cs	This manages pathways away from the lobby.
NextLevelManager.cs	This manages scene transitions.
PlayerController.cs	This is for controlling the player.
RespawnManager.cs	This is for respawning both enemies and the player.
NLGManager.cs	This is for holding the lexicon and running the syntactic component. It is for almost everything relevant to Natural Language Generation.
Noun.cs	This is the noun class to be used by the NLGManager script.
Verb.cs	This is the verb class to be used by the NLGManager script.

The game begins truly with the movement script. There is a physics engine inherent to Unity; the game designer determines specifics like the magnitude of gravity and whether velocity functions or force functions move an object. The movement script tackles these specifics: If the user presses "left," have the player move left with the movement speed (likewise with right). If the player presses "up," have the player jump with an upward impulse. If the player presses "left" in mid-air, exert a force on the player

to the left (likewise with right). If the player presses "shift," have the player move faster. If the player presses the spacebar, have the player attack. These conditionals are at the essence what the script does.

The next script is the enemy script. This script allows for enemies to move back and forth across platforms by giving them a range. Whenever an enemy reaches the end of its range, it turns to face the opposite direction. If the player approaches, the enemy tries to attack the player. Otherwise, enemies are of little complexity, and they are not as central to the game as movement. The game is more of a series of agility puzzles than a beat-em-up sort of game.

The respawn manager script controls respawning for both the player and the enemies. When the player dies, not only does the game reset the player's position to the checkpoint; enemies come back to life, too.

The final few scripts are not too involved. They are for: switching levels, choosing to go to different locations from a lobby, controlling sound, maintaining a constant aspect ratio across different screens, and saving the game.

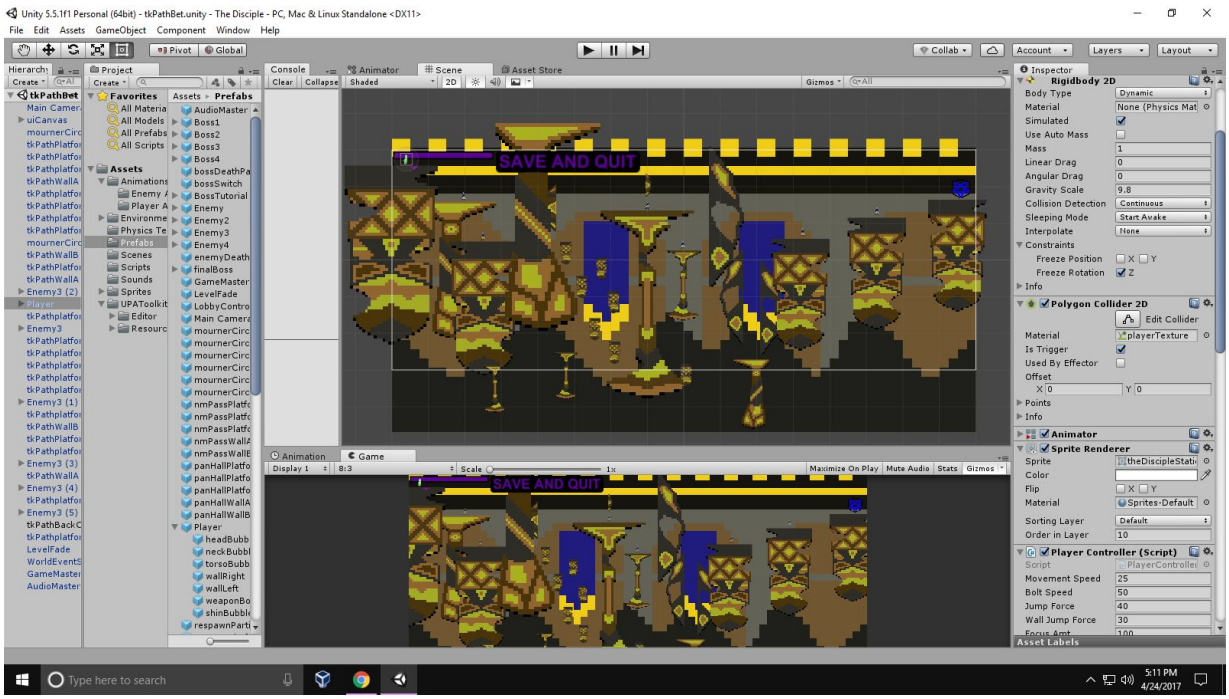
In a lot of platformers, movement is fairly simple. Players move at a constant, slow pace, and jumps are as high and as slow as jumps on the moon. With my game, I tried to make movement a little faster and more complex. While the movement mechanics are still unrealistic in the Disciple, moving is more dangerous and frenetic than in other platformers. Gravity is more punishing, but the Disciple can also jump up and away from walls. As unforgiving as the arrangement of platforms is, the player has the tools to move from the beginning to the end of the level. The movement mechanic is

closer to that of the modern platformer, *N*, than anything else. *N* is a very simple game wherein the player needs to complete several obstacle courses within a time limit by using the main character's quick movement and climbing abilities. Playing *The Disciple* is also about agility; it is a heavy challenge of one's dexterity.

Scripts aside, projects exist within Unity's interface as collections of scenes, where a scene is either a level or a menu. The game designer is always accessing one scene and working within that scene. In a scene, a game designer can create Game Objects or drag into the scene pre-existing Game Objects from the Project pane in Unity's interface. A Game Object is any discrete entity in the scene. It can be anything from a coin on the ground to a source of wind. For example, in my project, there are platforms, enemies, and walls. These are all Game Objects. When there is a pre-existing Game Object in the Project pane, that Game Object is a Prefab.

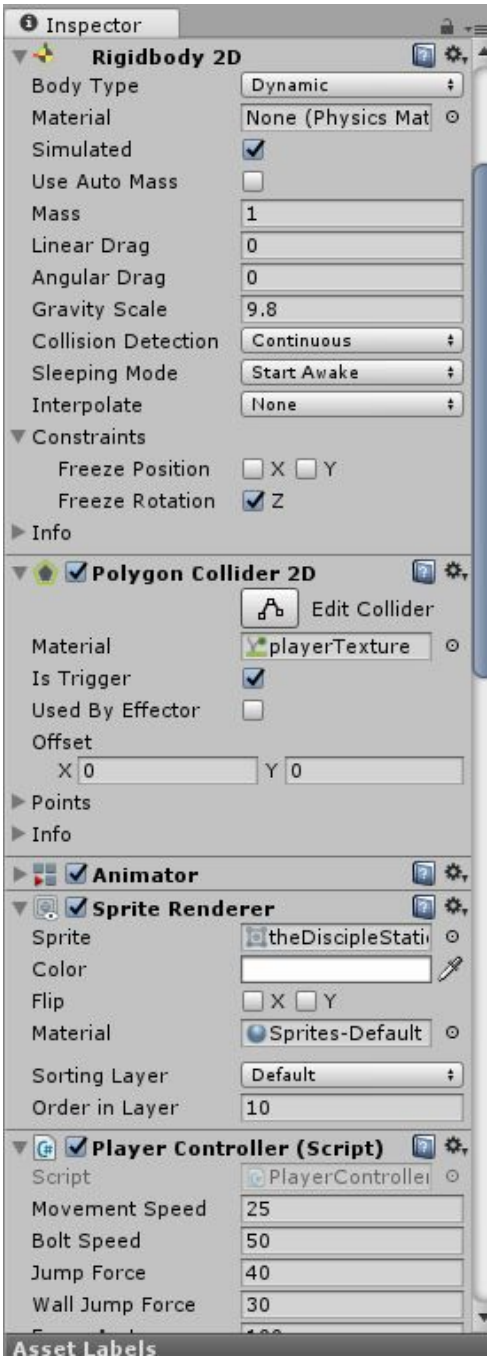
A Prefab is a Game Object with all of that Game Object's necessary components. A designer can drag Prefabs into the scene without making any significant changes to the game. For example, by creating a white circle (a default Game Object), flattening that circle into more of an ellipse, making the ellipse yellow, and dragging the ellipse into the project pane, a designer can create a "coin" Prefab that is ready to go into the scene numerous times. Otherwise, one would have to make several coins by instantiating white circles and making them look like coins again and again. Prefabs are half the project. Without them, level design is not possible. They are the building blocks with which to make the level. In *The Disciple*, there are more than a hundred, most of which are pieces of the environment, like walls or platforms. There is also a Prefab for the player, and

there are Prefabs for enemies. The most important components of these Prefabs in *The Disciple* are Rigid Bodies, which allow for Unity's physics engine to act upon the Game Object, Colliders, which keep track of what the Game Object is touching or prevent other Game Objects from passing through the Game Object, and Sprite Renderers, the reasons for which players can see anything happen in the game at all. Sprite Renderers allow for objects to be visible. The above scripts are also components of the Prefabs (e.g, PlayerController.cs, which is responsible for movement, is a component of the Player Prefab).



Each blue block in the left part of the screen is a Prefab that I can drag into the scene. The scene is visible in the large, central window. The rightmost pane displays each component that is attached to either a Game Object or a Prefab. This depends on what one selects in the Hierarchy or the Project pane. In the above example, I have selected

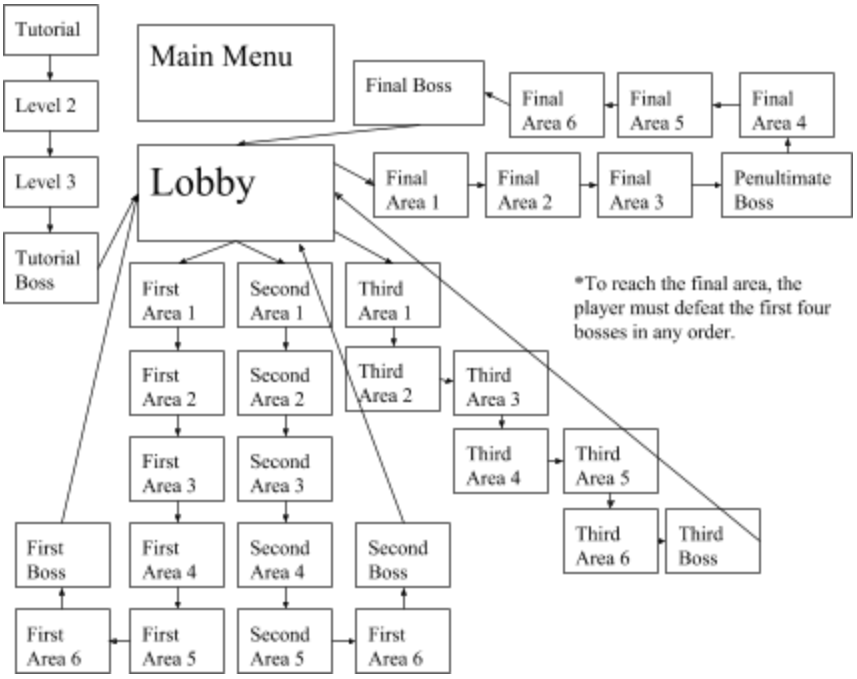
the Player, whose components include the Player Controller, a Rigid Body, a Collider, and a Sprite Renderer. These components are visible in the Inspector pane. Virtually all level design occurs within this interface. This is the right side of the screen, in detail.



Note that one can change the magnitude of gravity via the Rigid Body component. I have set the magnitude to 9.8 because I want the gravity to be realistically punishing. In addition, by accessing the Rigid Body component, one can freeze the rotation of the object so that it does not rotate, physics aside. One may set the collision detection to "continuous" instead of "discrete," so that the Game Object is more sensitive to collisions. Otherwise, Game Objects may pass through each other when they should not. Also, by using the interface, one can change directly the values of public variables in any script that is a component of a Game Object. Some of them are visible at the bottom of the figure, including movement speed, the velocity at which the player moves, and jump force, the force with which the player leaps into the air. This is the left side of the screen, in detail.

When my Prefabs were complete, the level design that remained was fairly easy. It was only a matter of dragging everything into the scene. The process would begin typically with my placement of the checkpoint and the goal, then I would move onto placement of the platforms. Finally, I would place the enemies, giving each of them their ranges and determining their movement velocities.

In each of these scenes, the player uses both agility and a contextual camera of the whole level to move from the checkpoint to the goal. If there is a boss in the scene, then the player must kill the boss in order to access the goal. These scenes come together to yield the following graph.



In terms of gameplay, I wanted to make something that was exceptionally challenging given the controls, so I pushed each level to the brink of what I thought was possible. In this game, the story is the struggle, itself. The fact that the Disciple, this deity who tries to become king, dies again and again, teleporting to the checkpoint upon death, is what demonstrates the tenacity of both the main character and the player. Ultimately, the Natural Language Generation component is what maintains this tenacity; what moves the game forward. Since deities transcend time in most mythologies, the Disciple embodies constantly the act of becoming king because either one is playing the game or one has played the game and is playing the game, again. The struggle is as constant as this deity's existence. Hardship is the theme I attempt to highlight in each of my narrative decisions; it is also what is behind the enemies and characters I have made with the sprite editor, some of which I show, here.

This is the man of the hour.

This is the vilest of imps. How dare he smile.

This is the boss no one wants to fight.

The inspiration behind this narrative is heavily from the three-dimensional action-role-playing series, *Dark Souls*. In each of the series' games, the player creates a main character, an Undead, who is cursed by the Darksign on his or her back. Every time an Undead dies or is killed by an enemy, this magical Darksign resurrects the player at the last checkpoint, a Bonfire. The curse is one of immortality, essentially. There is a fairly cut-and-paste medieval theme. The player needs to fight a lot of monsters one might see in The Lord of the Rings, like dragons and wraiths, in order to reach the lair of some cruel despot and displace the despot as the rightful owner of the throne.

The fantastical settings and beasts are not the point, though. What yields acclaim from the gaming community for *Dark Souls* is actually that it does *not* bog itself down with "world-building," a Cardinal Sin in the realm of genre fiction that leads to bad writing and, more importantly, earth-shatteringly intense boredom. The point is that both

the mechanics of the game and the simple behaviors of the characters ("behavior" in terms of what they do on and off the battlefield; not only in terms of what they say) tell the story of this medieval realm. Each beast and character represents a vague idea, with futility and hopelessness being common themes, since the game is incredibly challenging. For example, in *Dark Souls 2*, the king has descended into madness because of the monotony caused by the Darksign, but the game does not say this outright with any kind of text or speech. Most figures in the series representing both old age and high status have simply gone mad, and the fact that the Darksign has brought them to life several times is obviated by the grotesque appearances of their bodies (the Undead look like humans, but, as death triggers the Darksign again and again, they look more like zombies). *Dark Souls* is a prime example of, "Show, don't tell," and the fact that it is challenging makes it a fun game even without the story. I found it so artfully designed across the board that I attempted to do something similar, myself, in the form of a platformer.

What became most curious to me about my game's narrative was its absence of cutscenes. I had hoped to have them, at first, but, ultimately, I thought that focusing on them would have been a waste of time, since the game, together with the NLG component, tells the story, anyway. Most high-budget games have cutscenes so that there may exist a story in the world of the game, but the narrative scope of my game seemed so small that I thought cutscenes would not have made the game much better. As with *Dark Souls*, *The Disciple* tries to tell a story by drawing a picture. The pristine palace at the end of the game fails to have any kind of cracks in its platforms because the player has

entered the intimidating abode of perfection. The bosses are embodiments of isolation, rage, and fear. The game is the story.

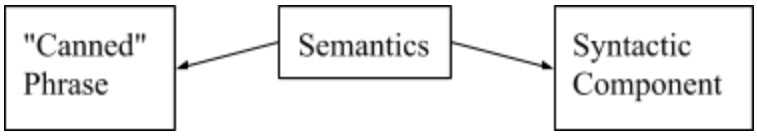
Once the game was complete, my debugging consisted of several play-throughs during which I would determine what work remained. I would edit accordingly. The rest of the game is Natural Language Generation.

Natural Language Generation in Unity

The NLG system I have created has its own script that may exist independently of the rest of the game. In fact, when I began construction of my NLG module, I wrote it in Python without attaching it to Unity. Ultimately, I abandoned this Python code and built the NLG module in C#. As the game exists currently, every other script is not necessarily relevant to NLG, and the game would still be able to function without the NLG system. The only task tying the NLG system to the other scripts is information retrieval. When something happens in-game for which a non-NLG script is responsible, one of the Boolean variables or numerical variables turns to a different value. Every frame, the NLG component poses several questions to the other scripts: Has the player fallen? Has the player been stabbed by an enemy? Has the player killed an enemy? These are a few examples. Scripts answer the NLG component by setting values, and once the values are set, one of two events happens. Given by the game's environment the current frame's combination of all of the variables, the system decides randomly between sending a "canned" phrase ("canned," meaning a complete sentence I wrote manually) and constructing a sentence by connecting the semantics to a syntactic component. The syntactic component uses a grammar that relates directly the variables to strings for individual words. The game may yield, for example, the following table.

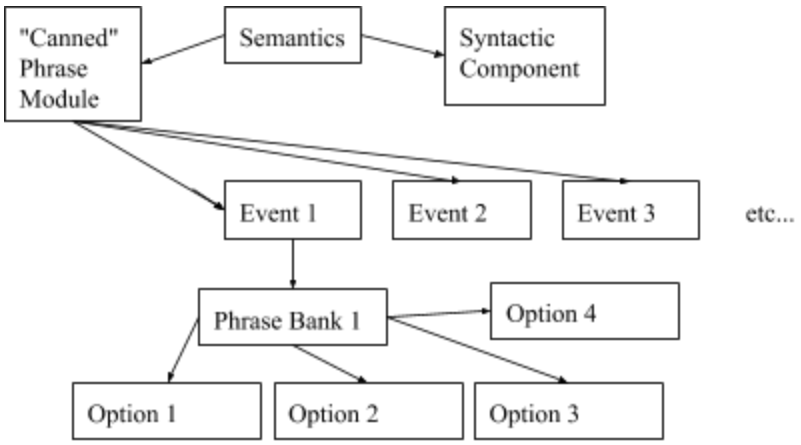
Events versus Sentences	Event Trigger: falling near the beginning of the level	Event Trigger: being killed by an enemy	Event Trigger: killing an enemy
Trial 1 Output	Deal with the drops to deal with the bad guys. (canned)	a killer has stabbed the Disciple. (generated)	I see, you like a good kebab. (canned)
Trial 2 Output	a drop kills the Disciple. (generated)	a killer obliterates the Disciple. (generated)	Bonk. (canned)
Trial 3 Output	You're off to a good start (canned)	a killer stabs the Disciple. (generated)	Tango down. (canned)

These two different paths exist in order to allow for greater variety in terms of what the NLG system can say. Whether a canned phrase or the syntactic component is used by the NLG system, both routes are informed by the semantics; the input.



What makes my set of canned phrases different from a typical phrase bank is that there are multiple inputs and multiple outputs. Instead of one event triggering a random selection among twenty phrases, there are about thirty events (each event is a set of value combinations) that trigger a random selection among four phrases I have written for each combination. Randomness remains because human beings do not usually say the same

thing twice, but the choice among all of the canned phrases is still informed by specific events that occur in-game.

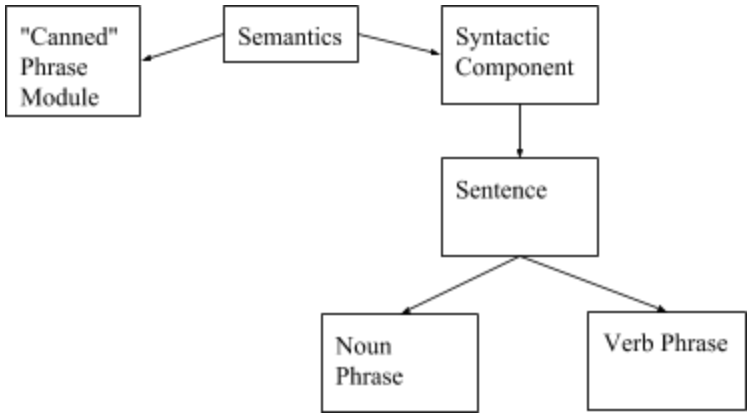


*The system never chooses the same phrase from a phrase bank twice in a row...

The syntactic component is a little more complicated. For each part of speech there is an array of strings, and each of an array's strings is one word. However, with respect to nouns and verbs, the NLG system employs a noun class and a verb class in order to account for nouns and verbs having different forms. For example, the noun class contains two strings, singular and plural, so that the NLG system can access both the singular form and the plural form of the same noun.

Next, this syntactic component decides what words to use. It decides what will be the subject, verb tense, and object of the sentence. It does this through both consideration of the values and random selection. Again, a human being rarely says the same thing

twice. Randomness is also used by the system for deciding whether to use pronouns and whether to use articles. Ultimately, there is a noun phrase, and there is a verb phrase. Both are the fundamental building blocks of what the system considers to be a sentence. The noun phrase becomes equal to whatever the subject is, and, most of the time, the singular form of the noun is chosen by the system. Similarly, the verb phrase becomes equal to whatever the object is, and the NLG system chooses the correct verb to place into the verb phrase. Finally, the sentence variable is set to the noun phrase plus the verb phrase (in that order). "The Disciple kills an enemy." "The enemy has obliterated the Disciple." etc. These are simple sentences wherein the subject, verb, and object are apparent enough for the sentences to be generated by the syntactic component. Whether the NLG system offers a canned phrase or feeds the set of variables through my grammar of "if" statements, there is enough diversity to address many different situations, and this diversity is at the essence of NLG: Many different inputs ought to yield many different outputs. There should be a lot of good connections between what happens and what the NLG system says.



The variables I use are in the appendix, but what may elucidate presently the relationship between the game's world and the NLG component is a compilation of my variables (some of which the current program does not use) in the following table. Booleans are for any piece of information that fails to have a scale, like the determination of whether a player has fallen, and numbers are for any piece of information that does involve scale. For each numerical variable, there is a conditional somewhere in the NLG script such that the script checks whether the number is either greater to or less than a certain value. The script provides a sentence relating to a numerical threshold.

Variable Name	Type	Measurement	Question
`hasFallen`	Boolean	Determine whether the player has fallen out of bounds	"Has the player fallen?"
`hasBeenStabbed`	Boolean	Determine whether the player has been killed by an enemy	"Has the player been stabbed?"
`hasStabbed`	Boolean	Determine whether the player has killed an enemy	"Has the player killed an enemy?"
`quickSuccessionFalls`	Boolean	Determine whether the player has fallen three times in quick succession	"Is the player falling excessively?"
`quickSuccessionStabs`	Boolean	Determine whether the player has been stabbed by enemies three times in quick succession	"Is the player being stabbed by enemies excessively?"
`hasKilledBoss`	Boolean	Determine whether the player has killed a boss	"Has the player killed a boss?"
`boltMeterEmpty`	Boolean	Determine whether the player's speed resource is empty	"Has the player enough energy to bolt?"

boltUnusedForLong	Boolean	Determine whether the speed boost ability has not been employed by the player for long	"Has it been a long while since the player has used bolt?"
aerialKill	Boolean	Determine whether the player has executed an aerial kill	"Has the player performed an aerial kill?"
deathsDuringLevel	Int	Determine the number of deaths the player has encountered over the course of a single level	"How many times has the player died during this level?"
deathsDuringGame	Int	Determine the number of deaths the player has encountered over the course of the whole game	"How many times has the player died during this game?"
killsDuringLevel	Int	Determine the number of kills the player has made over the course of a single level	"How many times has the player killed an enemy during this level?"
killsDuringGame	Int	Determine the number of kills the player has made over the course of the game	"How many times has the player killed an enemy during this game?"
inAir	Int	Determine how long the player has been in air	"How long has the player been in the air?"
distFromGoal	Float	Determine the distance between the player and the goal	"How far is the player from the goal?"

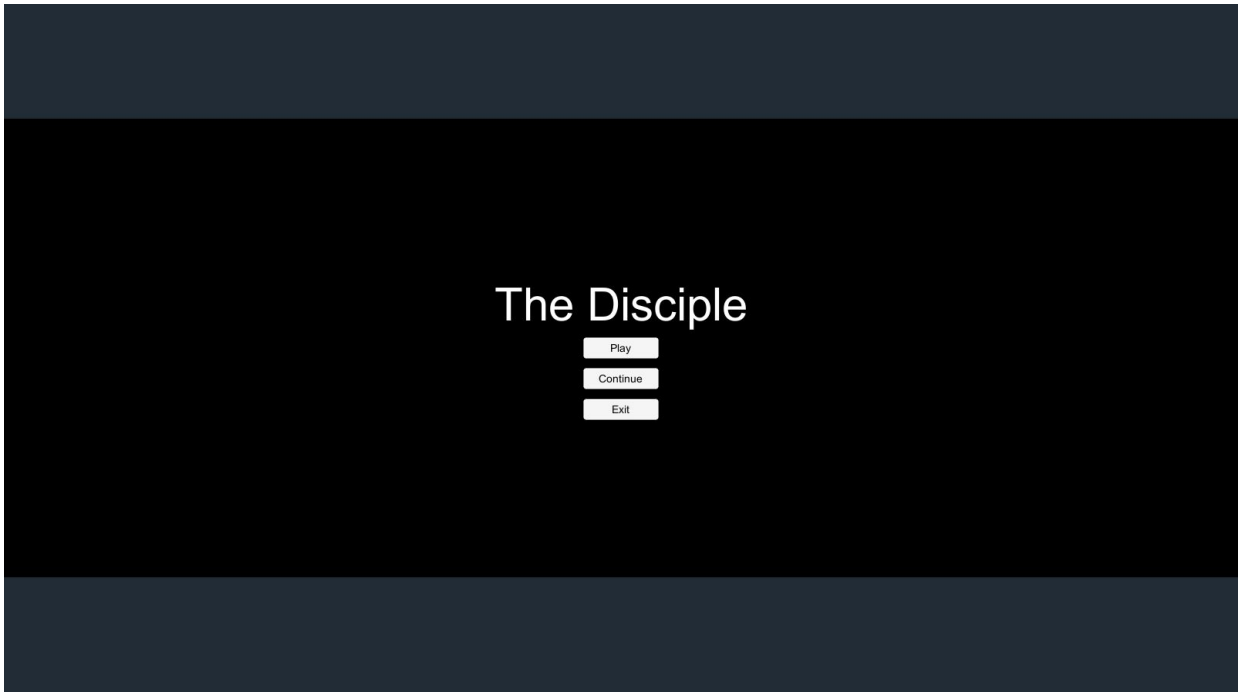
As for the combinations of variables, many of the drawings among my notes accumulate into the following pseudocode that includes only variables being used by the current NLG system.

```
hasFallen AND deathsDuringLevel % 60 AND deathsDuringLevel >= 60

hasFallen AND deathsDuringLevel % 10 AND deathsDuringLevel >= 10

hasFallen AND boltUnusedForLong

hasFallen AND inAir > 55

hasFallen AND distFromGoal > 125

hasFallen AND 60 < distFromGoal < 125

hasFallen AND distFromGoal < 60

hasBeenStabbed AND boltMeterEmpty

hasBeenStabbed AND deathsDuringLevel % 60 AND deathsDuringLevel >= 60

hasBeenStabbed AND deathsDuringLevel % 10 AND deathsDuringLevel >= 10

hasBeenStabbed AND boltUnusedForLong

hasBeenStabbed AND inAir > 55

hasBeenStabbed AND distFromGoal > 125

hasBeenStabbed AND 60 < distFromGoal < 125

hasBeenStabbed AND distFromGoal < 60

hasStabbed AND boltMeterEmpty

hasStabbed AND boltUnusedForLong

hasStabbed AND deathsDuringLevel % 10 AND deathsDuringLevel >= 10

quickSuccessionFalls AND boltMeterEmpty
```

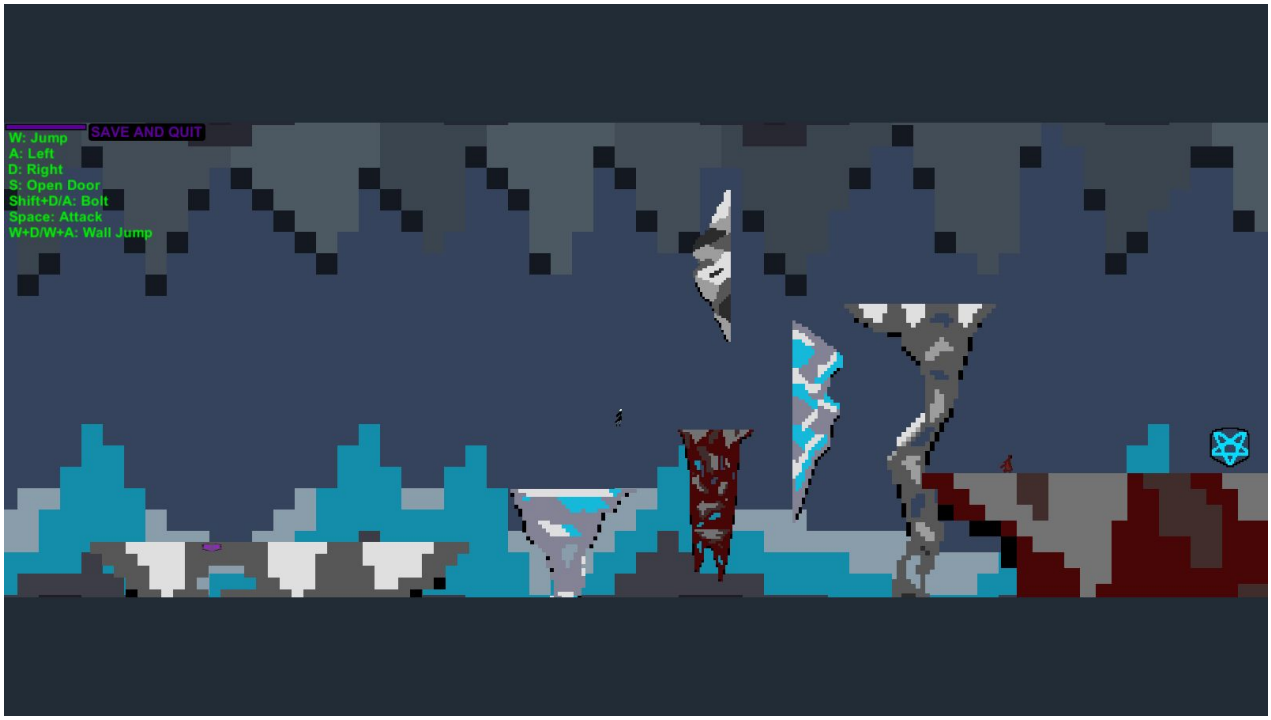
```
quickSuccessionFalls AND boltUnusedForLong

quickSuccessionFalls AND inAir > 55

quickSuccessionFalls AND distFromGoal > 125

quickSuccessionFalls AND 60 < distFromGoal < 125

quickSuccessionStabs AND boltMeterEmpty

quickSuccessionStabs AND boltUnusedForLong

quickSuccessionStabs AND distFromGoal > 125

quickSuccessionStabs AND 60 < distFromGoal < 125

hasKilledBoss AND boltUnusedForLong

hasKilledBoss AND aerialKill
```

Discussion

Ultimately, the NLG component functions as well as any other part of the game. With a little blurb at the top of the screen, the NLG system says something about what has happened or what the player is doing. This is what the platformer actually looks like, in action:

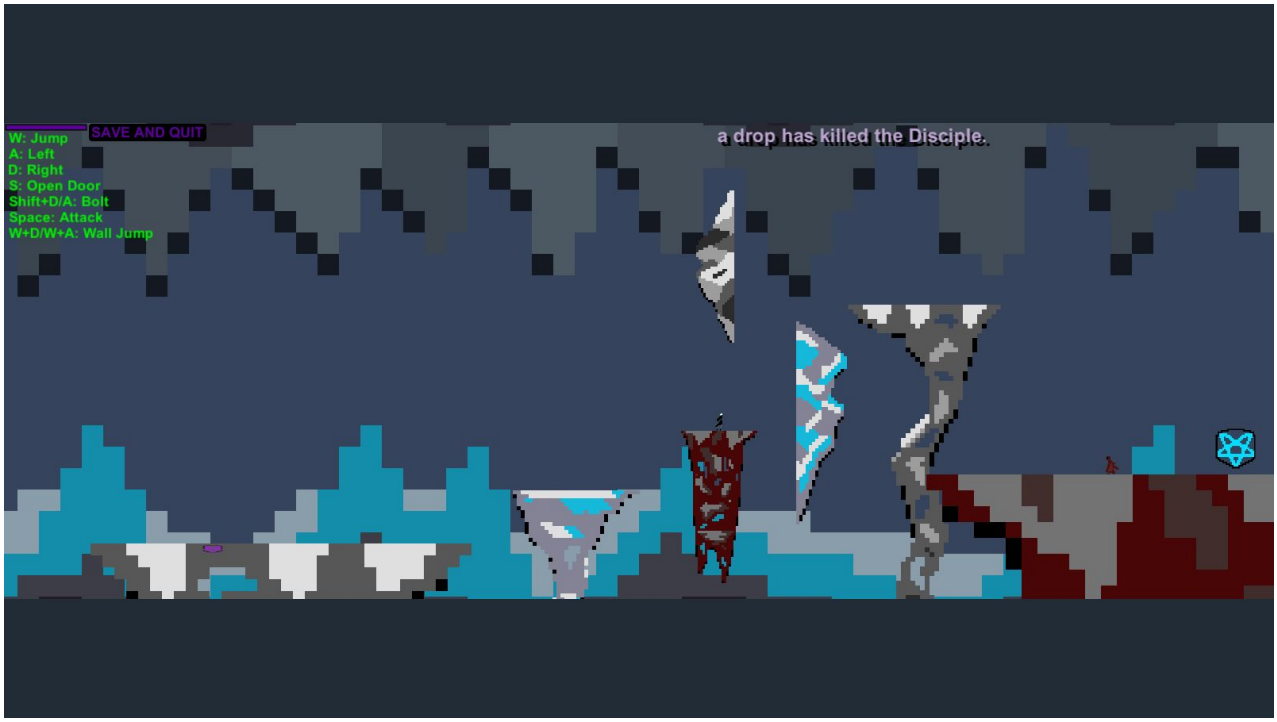


This is simply the main menu. What most levels look like follows.



(The Disciple is in mid-air at the center of the screen)

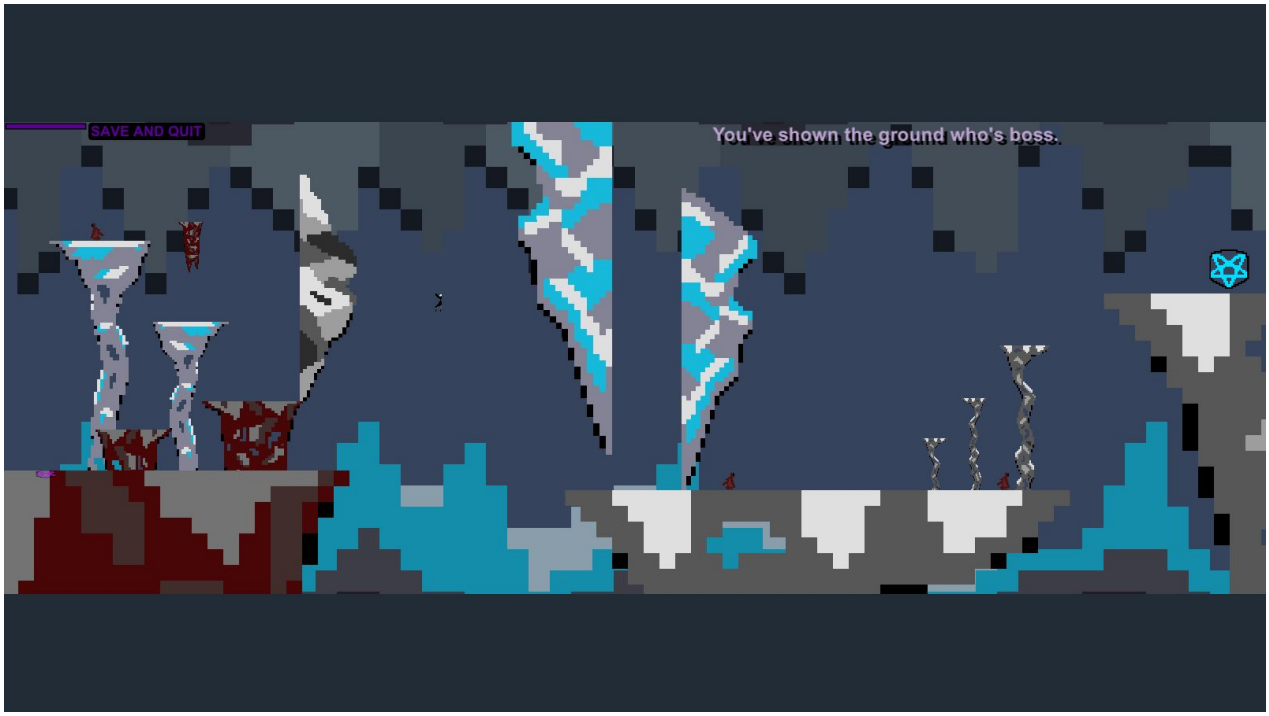
This is from the tutorial level. One can see the Disciple in the center of the screen, though there is no visible output from the NLG system.



At the top of the screen, a sentence constructed by the syntactic component comes into view. This does not happen until the player falls out of bounds at the bottom of the level. Next is an image that includes one of my canned phrases. The context is the same; the player has fallen.

(The Disciple is near the beginning of the level on the left)

As one can see, the canned phrases appear to be a bit more natural and complex.



(About a third of the way through the level, the Disciple is in mid-air)

In this next panel, I have killed an enemy. This yields:

In the next panel, I am fighting a boss. In this game, bosses are enemies whose complexity is only a matter of hitting the weak spot. Recall that the enemies are not of great complexity in this game.

The NLG system seems to be a nice addition to the game that adds some character. The following table displays a set of random events and the sentences to which those events lead.

Event	**Sentence**
Killing an enemy	Well, aren't you a foolish samurai warrior wielding a magic sword? (canned)
Killing an enemy	The Disciple stabs a killer. (generated)
Falling near the beginning	If you can't deal with the drops, you can't deal with the bad guys. (canned)
Being killed by an enemy at the beginning	That's too bad. (canned)
Being killed by an enemy at the beginning	Shame. (canned)
Killing an enemy	The Disciple has obliterated a killer. (generated)
Falling in the middle of the level	a drop kills the Disciple. (generated)
Falling in the middle of the level	You were doing kind of well. (canned)

Still, the system feels a bit limited. Some value combinations are clearly more common than others. This creates unnatural repetition, so, if I had another few months, I would think about the probability of combinations, having the NLG system talk about events that are more likely to happen instead of ones that fail to happen often. In fact, for the sake of more diversity in any case, I would put in more variables so that the NLG system would have more information to consider and more phrase banks from which to choose a statement. If I were to put in more variables, I would have to consider more combinations, so I would need inevitably to write more canned phrases.

The NLG system also does not even use some of its lexicon, though having access to too many words is better than having access to too few words. Considering the whole lexicon, I could embrace the tedium of thinking about what each specific word can do in a specific context. Finally, I could expand the syntactic component to offer imperative sentences or questions, since, at this point, the NLG system offers only simple sentences. This expansion would involve giving more rules to my grammar of "if" statements. While I know not what would make an imperative sentence more useful than a simple sentence in a specific context, I could at least allow the NLG system to have a random choice among different types of sentences in order to have the system appear more organic.

Where the current project stands is satisfactory to me. While it is not the latest and greatest MMORPG, it seems to open a lot of doors toward other projects. I can

always reuse the code I have created, already, and apply scripts to several other contexts. I may always elaborate upon the NLG system in the ways I have mentioned, as well.

Bibliography

Allen, James. Natural Language Understanding. Redwood City, CA: Benjamin/Cummings, 1995.
Print.

Bayarappu, Hemanth Sagar. "Efficient Algorithm for Context Sensitive Aggregation in Natural Language Generation." (2011): 84-89. Web. 17 Oct. 2016.

Dimitromanolaki, Aggeliki, and Ion Androutsopoulos. "Learning to Order Facts for Discourse Planning in Natural Language Generation." (n.d.): n. pag. Web. 17 Oct. 2016.

Funge, John David. Artificial Intelligence for Computer Games. Wellesley: K Peters, 2004. Print.

Reiter, Ehud, and Robert Dale. "Building Applied Natural Language Generation Systems." Natural Language Engineering 1.1 (1995): n. pag. Web. 16 Oct. 2016.

Reiter, Ehud, Somayajulu G. Sripada, and Roma Robertson. "Acquiring Correct Knowledge for Natural Language Generation." Journal of Artificial Intelligence Research 18 (2003): 491-516. Web. 19 Oct. 2016.

Sripada, Somayajulu G. "A Two-stage Model for Content Determination." (n.d.): n. pag. Web. 17
Oct. 2016.

Stent, Amanda. Natural Language Generation in Interactive Systems. Cambridge: Cambridge U, 2014. Print.

Stern, Andrew, and Michael Mateas. "*Facade*: An Experiment in Building a Fully-Realized Interactive Drama." Game Developers Conference (2003): n. pag. Web.

Strong, C., Manish Mehta, and Kinshuk Mishra. "Emotionally-driven Natural Language Generation for Personality-rich Characters in Interactive Games." In Proceeding Of: Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference (2007): 98-100. Web. 2 Oct. 2016.

Wardrip-Fruin, Noah. Expressive Processing: Digital Fictions, Computer Games, and Software Studies. Cambridge, MA: MIT, 2012. Print.

Appendix

For reference and general explanation, the following scripts are the ones I have made during this project. They contain the lexicon, the phrase banks, and the syntactic component. Most lines are explained by a comment.

PlayerController.cs

```
1.  using UnityEngine.SceneManagement;
2.  using UnityEngine.UI;
3.  using System.Collections;
4.
5.  public class PlayerController : MonoBehaviour
6.  {
7.      //REMEMBER ENVIRONMENT LABELS, TAGS, LAYERS, ETC.
8.
9.      [SerializeField]
10.     private float movementSpeed = 25f; //This is the speed at which the player moves.
11.
12.     [SerializeField]
13.     private float boltSpeed = 50f; //This is the speed at which the player moves while the
    player holds the "shift" key.
14.
15.     [SerializeField]
16.     private float jumpForce = 40f; //This is the vector with which the player jumps.
17.
18.     [SerializeField]
19.     private float wallJumpForce = 30f; //This is the vertical vector with which the player leaps
    from a wall.
20.
21.     [SerializeField]
22.     private float focusAmt = 100f; //This is the maximum amount of the resource responsible
    for the player's ability to bolt.
23.
24.     public Animator playerAnim; //This is the animator responsible for the player's animations.
25.
26.     [SerializeField]
27.     public GameObject wBox; //"W" is for weapon.  This box allows for the program to
    determine whether the player's sword has connected with an enemy.
28.
29.     [SerializeField]
30.     private LayerMask gnd; //This is a reference to the "ground" layer.
31.
```

```
32.     [SerializeField]
33.     private LayerMask walls; //This is a reference to the "walls" layer.
34.
35.     private RectTransform boltBar; //This UI component is a visualization of how much focus
    the player has.
36.
37.     public Rigidbody2D playerRgdBdy; //This is a reference to the RigidBody component of the
    "player" GameObject.
38.
39.     private CircleCollider2D wallL; //This is a reference to the collider responsible for detecting
    whether there is a wall to the player's left.
40.
41.     private CircleCollider2D wallR; //This is a reference to the collider responsible for detecting
    whether there is a wall to the player's right.
42.
43.     private CircleCollider2D groundCircle; //This is a reference to the collider responsible for
    ground detection.
44.
45.     private bool dirBool = true; //This Boolean is responsible for changing the direction of the
    player.
46.
47.     private AudioMaster audioMast; //This is the script responsible for any in-game noises the
    player makes.
48.
49.     //The following three strings are for referencing specific sounds.
50.     public string jumpNoise = "jump";
51.     public string landNoise = "groundHit";
52.     public string deathNoise = "playerDeath";
53.
54.     private bool isGrounded; //This boolean determines whether the player is on the ground.
55.
56.     private NLGManager nlgMan;
57.
58.     private float boltUnusedForLongTime = 0;
59.
60.     void Start()
61.     {
62.         //All statements in this method set the variables to the correct components of the correct
    GameObjects.
63.         boltBar = GameObject.Find("boltBar").GetComponent<RectTransform>();
64.         playerRgdBdy = GetComponent<Rigidbody2D>();
65.         wallL = GameObject.Find("wallLeft").GetComponent<CircleCollider2D>();
66.         wallR = GameObject.Find("wallRight").GetComponent<CircleCollider2D>();
67.         groundCircle = GetComponent<CircleCollider2D>();
68.         nlgMan = GetComponent<NLGManager>();
69.         audioMast = AudioMaster.instance;
70.     }
71.
72.     void Update() //This built-in method runs during each frame.
73.     {
74.
75.         boltUnusedForLongTime++;
76.         if (nlgMan.hasStabbed && !groundCircle.IsTouchingLayers(gnd))
77.         {
78.             nlgMan.aerialKill = true;
79.         }
80.         if (boltUnusedForLongTime > 10000)
81.         {
82.             nlgMan.boltUnusedForLong = true;
83.         }
```

```
84.         if (groundCircle.IsTouchingLayers(walls) && !wallL.IsTouchingLayers(walls) &&
    !wallR.IsTouchingLayers(walls)) //If the groundCircle touches the walls and neither wallL nor
    wallR touches the walls, perform the following.
85.         {
86.             playerRgdBdy.AddForce(new Vector2(0, jumpForce/4), ForceMode2D.Impulse);
    //Execute a small leap so that the player may not slide along horizontal segments of the wall.
87.             return; //Ensure that there are no especially high leaps.
88.         }
89.         if (Input.GetKeyDown(KeyCode.Space) && !playerAnim.GetBool("isAttacking")) //If the
    player has attacked and is not attacking, already, perform the following.
90.         {
91.             playerAnim.SetBool("isSliding", false); //Stop the wall slide animation if the player is
    attacking.
92.             StartCoroutine(Strike()); //Begin the Strike( ) function.
93.         }
94.         boltBar.sizeDelta = new Vector2(focusAmt, 5); //Change the size of the boltBar given the
    focusAmt.
95.         movementAerial(); //Execute the movementAerial( ) function.
96.         if (playerRgdBdy.velocity.y < -5) //If the player's y-component velocity is less than -5
    m/s, perform the following.
97.             playerAnim.SetBool("isFalling", true); //Begin the falling animation.
98.         else
99.             playerAnim.SetBool("isFalling", false); //Perform the jumping animation, instead.
100.        }
101.
102.        void FixedUpdate() //This built-in method runs during each frame, but the interval
    between two of the method's calls is static.
103.        {
104.            bool wasGrounded = isGrounded; //This reference is for sound management, only.
105.            isGrounded = false; //By default, isGrounded is equal to false during every frame.
106.            movementGnd(); //Execute the movementGnd( ) function.
107.            if (playerRgdBdy.velocity.x < -0.3 && dirBool == true) //If the player moves to the
    left and dirBool is true, perform the following.
108.                changeDir(); //Change the direction in which the player faces.
109.            else if (playerRgdBdy.velocity.x > 0.3 && dirBool == false) //If the player moves to
    the right and dirBool is false, perform the following.
110.                changeDir(); //Change the direction in which the player faces.
111.            if (wasGrounded != isGrounded && isGrounded == true) //If the player has touched
    the ground during an instant, perform the following.
112.            {
113.                audioMast.PlaySound(landNoise); //Emit this sound via the "audio" script.
114.            }
115.        }
116.
117.        void changeDir() //Change the direction in which the player faces.
118.        {
119.            Vector2 playerScale = transform.localScale; //Reference the scale of the object so
    that the object may reverse.
120.            if (dirBool == true) //If the player is facing right, perform the following.
121.            {
122.                //Change the positions of the wallR, wallL, and the player, accordingly.
123.                transform.position = new Vector2(transform.position.x - 0.7f,
    transform.position.y); // Keep the player from moving when the player flips.
124.                //wallL must always be to the player's left, and wallR must always be to the
    player's right.
125.                //The two colliders must swap positions.
126.                wallR.transform.localPosition = new Vector2(-0.37f, -0.4f);
127.                wallL.transform.localPosition = new Vector2(-0.15f, -0.4f);
128.            }
129.            else
130.            {
```

```
131.             //Change the positions of the wallR, wallL, and the player, accordingly.
132.             transform.position = new Vector2(transform.position.x + 0.7f,
    transform.position.y); // Keep the player from moving when the player flips.
133.             //wallL must always be to the player's left, and wallR must always be to the
    player's right.
134.             //The two colliders must swap positions.
135.             wallR.transform.localPosition = new Vector2(-0.15f, -0.4f);
136.             wallL.transform.localPosition = new Vector2(-0.37f, -0.4f);
137.         }
138.         playerScale.x *= -1; //Invert the scale of the player.
139.         transform.localScale = playerScale; //Store the current scale of the player in the
    "playerScale" variable.
140.         dirBool = !dirBool; //Invert the 'dirBool' Boolean value.
141.     }
142.
143.     void movementAerial() //This method is responsible for the player's aerial movement.
144.     {
145.         if (!groundCircle.IsTouchingLayers(gnd) && (wallL.IsTouchingLayers(walls) ||
    wallR.IsTouchingLayers(walls))) //If the player is not touching the ground and either wallL or
    wallR touches a wall, perform the following.
146.         {
147.             playerAnim.SetBool("isSliding", true); //Begin the sliding animation.
148.         }
149.         else //By default, the player is neither jumping nor sliding.
150.         {
151.             playerAnim.SetBool("isJumping", false); //End the jumping animation.
152.             playerAnim.SetBool("isSliding", false); //End the sliding animation.
153.         }
154.         if (Input.GetKeyDown("w")) //If the player has pressed this key, perform the
    following.
155.         {
156.             if (groundCircle.IsTouchingLayers(gnd)) //If the player is touching the ground,
    perform the following.
157.             {
158.                 audioMast.PlaySound(jumpNoise); //Emit the "jumpNoise."
159.                 playerAnim.SetBool("isJumping", true); //begin the jumping animation.
160.                 playerRgdBdy.AddForce(new Vector2(0, jumpForce), ForceMode2D.Impulse);
    //Add an upward impulse.
161.             }
162.             else
163.             {
164.                 if (wallL.IsTouchingLayers(walls) && Input.GetAxis("Horizontal") != -1) //If
    wallL touches the wall and the player is not trying to move into the wall, perform the following.
165.                 {
166.                     audioMast.PlaySound(jumpNoise); //Emit the "jumpNoise."
167.                     playerRgdBdy.AddForce(new Vector2(30, wallJumpForce),
    ForceMode2D.Impulse); //Add an upward, sideways impulse away from the wall.
168.                 }
169.                 if (wallR.IsTouchingLayers(walls) && Input.GetAxis("Horizontal") != 1) //If
    wallR touches the wall and the player is not trying to move into the wall, perform the following.
170.                 {
171.                     audioMast.PlaySound(jumpNoise); //Emit the "jumpNoise."
172.                     playerRgdBdy.AddForce(new Vector2(-30, wallJumpForce),
    ForceMode2D.Impulse); //Add an upward, sideways impulse away from the wall.
173.                 }
174.             }
175.             return; //This forbids especially high leaps.
176.         }
177.     }
178.
```

```
179.        void movementGnd() //This method is responsible for movement while the player is on the ground, excepting the functionality after the initial "else."
180.        {
181.            if (groundCircle.IsTouchingLayers(gnd)) //If the player's circle collider is touching the ground, perform the following.
182.            {
183.                nlgMan.inAir = 0;
184.                //By default, if the player is on the ground, the player is not running, and isGrounded is true.
185.                playerAnim.SetBool("isRunning", false);
186.                isGrounded = true;
187.                if (Input.GetAxis("Horizontal") != 0) //If the player attempts to move, perform the following.
188.                {
189.                    playerAnim.SetBool("isRunning", true); //Begin the running animation.
190.                    playerRgdBdy.velocity = new Vector2(Input.GetAxis("Horizontal") * movementSpeed, playerRgdBdy.velocity.y); //Base the player's velocity on the player's input.
191.                    if (focusAmt >= 3 && (Input.GetKey(KeyCode.LeftShift) == true || Input.GetKey(KeyCode.RightShift) == true)) //If the player's amount of focus is greater than or equal to 3 and the player has pressed the "shift" key, perform the following.
192.                    {
193.                        nlgMan.boltUnusedForLong = false;
194.                        boltUnusedForLongTime = 0;
195.                        focusAmt -= 10; //Decrease the amount of focus, since the player is using it.
196.                        playerAnim.speed = 2; //Increase the speed of the animation, only.
197.                        playerRgdBdy.velocity = new Vector2(Input.GetAxis("Horizontal") * boltSpeed, playerRgdBdy.velocity.y); //Double the player's speed.
198.                    }
199.                    else if (focusAmt < 4)
200.                    {
201.                        nlgMan.boltMeterEmpty = true;
202.                    }
203.                    playerAnim.speed = 1; //Have the animator return to its original speed.
204.                }
205.            }
206.            else //If the player fails to touch the ground, perform the following.
207.            {
208.                nlgMan.inAir++;
209.                if (playerRgdBdy.velocity.x <= movementSpeed && playerRgdBdy.velocity.x >= -movementSpeed) //If the player's velocity is less than movementSpeed, perform the following.
210.                {
211.                    playerRgdBdy.AddForce(new Vector2(Input.GetAxis("Horizontal") * 30 * movementSpeed, 0f), ForceMode2D.Force); //Given input from the player, exert a force on the airborne player.
212.                }
213.                else //If the player's velocity is greater than movementSpeed, perform the following.
214.                {
215.                    playerRgdBdy.AddForce(new Vector2(-(playerRgdBdy.velocity.x * 1.5f), 0f), ForceMode2D.Force); //Exert a slight force on the player (the force is opposite the current movement) so that the player does not continue to accelerate.
216.                }
217.                //The player is not running, but the player is jumping.
218.                playerAnim.SetBool("isRunning", false); //End the running animation.
219.                playerAnim.SetBool("isJumping", true); //Begin the jumping animation.
220.            }
221.            if (focusAmt < 100) //If the player's focus is beneath full capacity, perform the following.
222.                focusAmt += 2; //Replenish focus.
223.        }
```

```
224.
225.        public IEnumerator Strike() //Perform an attack.
226.        {
227.            playerAnim.SetBool("isAttacking", true); //Begin the attack animation.
228.            yield return new WaitForSeconds(0.02f); //Wait for 0.02 seconds.
229.            wBox.SetActive(true); //Activate the weapon's collider, so that it may collide with an
    enemy.
230.            yield return new WaitForSeconds(0.14f); //Wait for 0.14 seconds.
231.            playerAnim.SetBool("isAttacking", false); //End the attack animation.
232.            wBox.SetActive(false); //Deactivate the weapon's collider.
233.        }
234.    }
```

Enemy.cs

```
1.  using UnityEngine;
2.  using UnityEngine.SceneManagement;
3.  using System.Collections;
4.
5.  public class Enemy : MonoBehaviour {
6.
7.      private NLGManager playerNlg;
8.
9.      [SerializeField]
10.     private float enemySpeed = 10f; //This is the speed at which the enemy moves.
11.
12.     [SerializeField]
13.     private float rangeRadius = 10f; //This is the distance to which the enemy walks before
    turning toward the other direction.
14.
15.     [SerializeField]
16.     public Animator enemyAnim; //This is the animator responsible for animating the enemy.
17.
18.     [SerializeField]
19.     public GameObject wBoxEnemy; //This is the enemy's weapon.
20.
21.     [SerializeField]
22.     private GameObject playerPrefab; //This is the player in the scene.
23.
24.     [SerializeField]
25.     private PlayerController playerScript; //This is the script responsible for controlling the
    player.
26.
27.     private RespawnManager respawnScript; //This is the script responsible for bringing the
    player and the enemies back to life.
28.
29.     public GameObject enemyDeathParticles; //This is the particleSystem that appears when
    the enemy dies.
30.
31.     [SerializeField]
32.     private GameObject exitPointBoss; //This is the exit of a boss level; such exits are hidden
    by the game until the player defeats the boss.
33.
34.     private Rigidbody2D enemyRgdBdy; //This is the RigidBody component of the enemy
    object.
35.
36.     public float AtkRangeVert = 3; //This is the vertical range within which the enemy initiates
    an attack.
37.     public float AtkRangeHor = 5; //This is the horizontal range within which the enemy
    initiates an attack.
```

```
38.
39.     [SerializeField]
40.     private bool isStandardBoss = false; //This Boolean keeps track of whether there is a
    typical boss in the scene.
41.
42.     [SerializeField]
43.     private bool isRwKeepBoss = false; //This Boolean keeps track of whether the current
    boss is Nittonio, if there is a boss at all.
44.
45.     [SerializeField]
46.     private bool isTfPalaceBoss = false; //This Boolean keeps track of whether the current
    boss is Qalem, if there is a boss at all.
47.
48.     //The following two variables regard the enemy's movement.
49.
50.     float RangeA;
51.     float RangeB;
52.
53.     //The following two variables allow for the relocation of the enemy if the enemy falls.
54.
55.     float SpawnX;
56.     float SpawnY;
57.
58.     //This is for flipping the enemy in the two-dimensional space.
59.
60.     float dir = 1; //Have the enemy face right, at first.
61.
62.     private AudioMaster audioMast; //This is the script in which one finds sound functionality
    for the enemy.
63.
64.     public string deathNoise = "enemyDeath"; //This string is a reference to the noise the
    enemy makes upon death.
65.
66.     //The following four Booleans are for determining when to attack the player.
67.
68.     private bool shouldAtkRight;
69.     private bool shouldAtkLeft;
70.     private bool shouldAtkUp;
71.     private bool shouldAtkDown;
72.
73.     void Start()
74.     {
75.         playerNlg = GameObject.Find("Player").GetComponent<NLGManager>();
76.         wBoxEnemy.SetActive(false);
77.         enemyRgdBdy = GetComponent<Rigidbody2D>();
78.         RangeA = transform.position.x + rangeRadius;
79.         RangeB = transform.position.x - rangeRadius;
80.         SpawnX = transform.position.x;
81.         SpawnY = transform.position.y;
82.         audioMast = AudioMaster.instance;
83.         respawnScript =
    GameObject.Find("mournerCircle").GetComponent<RespawnManager>();
84.     }
85.
86.     void Update()
87.     {
88.         //Find values for each boolean during an instant.  These booleans are for the enemies'
    "intelligence" functionality.
89.         shouldAtkRight = dir == 1 && playerPrefab.transform.position.x <= transform.position.x
    + AtkRangeHor && playerPrefab.transform.position.x >= transform.position.x;
```

```
90.         shouldAtkLeft = dir == -1 && playerPrefab.transform.position.x >= transform.position.x -
    AtkRangeHor && playerPrefab.transform.position.x <= transform.position.x;
91.         shouldAtkUp = playerPrefab.transform.position.y <= transform.position.y +
    AtkRangeVert;
92.         shouldAtkDown = playerPrefab.transform.position.y >= transform.position.y -
    AtkRangeVert;
93.         if (Physics2D.IsTouching(playerScript.wBox.GetComponent<BoxCollider2D>(),
    GetComponent<BoxCollider2D>()) || transform.position.y <-30) //If either the enemy has
    contacted the player's sword or has fallen, kill the enemy.
94.         {
95.             StartCoroutine(killEnemy());
96.             playerNlg.killsDuringLevel++;
97.             playerNlg.killsDuringGame++;
98.         }
99.         strikeAI();
100.            enemyMvmt();
101.        }
102.
103.        void enemyMvmt()
104.        {
105.            if (enemyRgdBdy.velocity.x <= 5 && enemyRgdBdy.velocity.x >= -5) //If the
    enemy's velocity is beneath a certain amount, end the running animation.
106.                enemyAnim.SetBool("isRunning", false);
107.            else //Otherwise, have the enemy run.
108.                enemyAnim.SetBool("isRunning", true);
109.
110.            if (transform.position.x <= RangeB) //If the enemy has not reached the movement
    radius in the positive direction, perform the following.
111.            {
112.                dir = 1;
113.                transform.rotation = new Quaternion(0f, 0f, 0f, 0f); //Have this be the current
    orientation of the enemy (Quaternions are responsible for rotation).
114.            }
115.
116.            if (transform.position.x >= RangeA) //If the enemy has not reached the movement
    radius in the negative direction, perform the following.
117.            {
118.                dir = -1;
119.                transform.rotation = new Quaternion(0f, 180f, 0f, 0f); //Have this be the current
    orientation of the enemy (Quaternions are responsible for rotation).
120.            }
121.            enemyRgdBdy.velocity = new Vector2(enemySpeed * dir, enemyRgdBdy.velocity.y);
    //The enemy moves in whatever direction is provided by the "dir" variable.
122.        }
123.
124.        public IEnumerator killEnemy()
125.        {
126.            wBoxEnemy.SetActive(false); //Deactivate the enemy's attack.
127.            enemyAnim.SetBool("isAttacking", false);
128.            enemyAnim.SetBool("isDying", true);
129.            yield return new WaitForSeconds(.08f); //Give the fading animation some time.
130.            if (transform.position.y >= -30)
131.            {
132.                playerNlg.hasStabbed = true;
133.            }
134.            enemyAnim.SetBool("isDying", false);
135.            gameObject.SetActive(false); //Deactivate the entire gameObject.
136.            if (isStandardBoss) //If the player is in a typical boss level, perform the following.
137.            {
138.                playerNlg.hasKilledBoss = true;
```

```
139.            deathNoise = "bossDeath"; //Use this noise instead of the standard enemy death
    noise.
140.            //Instantiate the boss's death particles, and destroy them after 6 seconds.
141.            GameObject otherClone = Instantiate(enemyDeathParticles, new
    Vector2(transform.position.x, transform.position.y - 18), new Quaternion(0f, 0f, 0f, 0f)) as
    GameObject;
142.            otherClone.transform.rotation = Quaternion.Euler(270, 0, 0);
143.            Destroy(otherClone, 6f);
144.            exitPointBoss.SetActive(true);
145.            GameMaster.bossCount++;
146.        }
147.        else
148.        {
149.            //Instantiate the enemy's death particles, and destroy them after 3 seconds.
150.            GameObject otherClone = Instantiate(enemyDeathParticles, new
    Vector2(transform.position.x, transform.position.y - 1), new Quaternion(0f, 0f, 0f, 0f)) as
    GameObject;
151.            otherClone.transform.rotation = Quaternion.Euler(270, 0, 0);
152.            Destroy(otherClone, 3f);
153.            if (isTfPalaceBoss && GameObject.FindWithTag("Enemy") == null) //If the player
    is in the final scene and there are no more enemies, perform the following.
154.            {
155.                exitPointBoss.SetActive(true); //Activate the exit so that the player may return
    to the lobby.
156.                GameMaster.bossCount++; //Increment the 'bossCount' variable.
157.            }
158.
159.        }
160.        transform.localPosition = new Vector2(SpawnX, SpawnY); //Have the enemy appear
    at its original location.
161.        audioMast.PlaySound(deathNoise);
162.    }
163.
164.    public IEnumerator enemyStrike()
165.    {
166.        if (isRwKeepBoss) //If the player is near the first boss, perform the following.
167.        {
168.            GetComponent<BoxCollider2D>().enabled = false; //Deactivate the weak area of
    the boss when the boss attacks.
169.        }
170.        enemyAnim.SetBool("isAttacking", true);
171.        yield return new WaitForSeconds(0.02f); //Give the animation some time.
172.        wBoxEnemy.SetActive(true);
173.        yield return new WaitForSeconds(0.14f); //Give the animation some time, and give
    the weapon box some time to catch the player.
174.        if (Physics2D.IsTouching(wBoxEnemy.GetComponent<BoxCollider2D>(),
    playerPrefab.GetComponent<PolygonCollider2D>())) //If the enemy's sword connects with the
    player's collider, perform the following.
175.        {
176.            wBoxEnemy.SetActive(false); //Don't kill the player multiple times.
177.            playerNlg.hasBeenStabbed = true;
178.            respawnScript.killPlayer(); //Kill the player.
179.
180.        }
181.        wBoxEnemy.SetActive(false); //Deactivate the enemy's weapon box once the
    enemy's finished attacking.
182.        enemyAnim.SetBool("isAttacking", false);
183.        if (isRwKeepBoss) //If the player is near the first boss, perform the following.
184.        {
185.            GetComponent<BoxCollider2D>().enabled = true; //Reactivate the weak area
    when the boss is no longer attacking.
```

```
186.         }
187.       }
188.
189.       void strikeAI()
190.       {
191.          if (shouldAtkUp && shouldAtkDown && (shouldAtkLeft || shouldAtkRight)) //If the
    player is within a certain area and the enemy is facing that area, attempt to hit the player.
192.          {
193.             StartCoroutine(enemyStrike());
194.          }
195.       }
196.    }
```

RespawnManager.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.SceneManagement;
5.  using UnityEngine.UI;
6.
7.  public class RespawnManager : MonoBehaviour {
8.
9.     private bool playerKilled;
10.
11.    private int quickSuccessionStabsCount;
12.
13.    private int quickSuccessionStabsTime;
14.
15.    private int quickSuccessionFallsCount;
16.
17.    private int quickSuccessionFallsTime;
18.
19.    private PlayerController playerScript;
20.
21.    private NLGManager playerNlg;
22.
23.    private AudioMaster audioMast;
24.
25.    private GameObject[] enemyGameObjects;
26.
27.    private Text deathTextAppear;
28.
29.    private GameObject spawnPoint;
30.
31.    [SerializeField]
32.    private GameObject respawnParticles;
33.
34.    void Start () {
35.       quickSuccessionFallsCount = 0;
36.       quickSuccessionFallsTime = 0;
37.       quickSuccessionStabsCount = 0;
38.       quickSuccessionStabsTime = 0;
39.       deathTextAppear = GameObject.Find("DeathTextAppear").GetComponent<Text>();
40.       audioMast = AudioMaster.instance;
41.       enemyGameObjects = GameObject.FindGameObjectsWithTag("Enemy");
42.       playerScript = GameObject.Find("Player").GetComponent<PlayerController>();
43.       playerNlg = GameObject.Find("Player").GetComponent<NLGManager>();
44.       spawnPoint = gameObject; //reference the spawn location
45.    }
```

```
46.
47.     void Update()
48.     {
49.         deathTextAppear.text = playerNlg.S;
50.     }
51.
52.     void LateUpdate () {
53.         quickSuccessionStabsTime++;
54.         quickSuccessionFallsTime++;
55.         if (playerNlg.hasBeenStabbed)
56.         {
57.             if (quickSuccessionStabsTime < 500)
58.             {
59.                 quickSuccessionStabsCount++;
60.             }
61.             if (quickSuccessionStabsCount >= 3)
62.             {
63.                 quickSuccessionStabsCount = 0;
64.                 playerNlg.quickSuccessionStabs = true;
65.             }
66.             quickSuccessionStabsTime = 0;
67.         }
68.
69.         if (playerScript.gameObject.transform.position.y < -30)
70.         {
71.             playerNlg.hasFallen = true;
72.             if (quickSuccessionFallsTime < 240)
73.             {
74.                 quickSuccessionFallsCount++;
75.             }
76.             if (quickSuccessionFallsCount >= 3)
77.             {
78.                 quickSuccessionFallsCount = 0;
79.                 playerNlg.quickSuccessionFalls = true;
80.             }
81.             quickSuccessionFallsTime = 0;
82.             killPlayer();
83.         }
84.     }
85.
86.     public void killPlayer()
87.     {
88.         playerNlg.deathsDuringLevel++;
89.         playerNlg.deathsDuringGame++;
90.         playerScript.wBox.SetActive(false); //Deactive the player's weapon collider.
91.         playerScript.playerAnim.SetTrigger("hasDied"); //Begin the death/fade animation.
92.         GameObject clone = Instantiate(respawnParticles, new
    Vector2(playerScript.gameObject.transform.position.x,
    playerScript.gameObject.transform.position.y - 1), new Quaternion(0f, 0f, 0f, 0f)) as
    GameObject; //Instantiate particles upon death.
93.         clone.transform.rotation = Quaternion.Euler(270, 0, 0); //Ensure correct direction of
    particle system.
94.         Destroy(clone, 2f); //Destroy the particle system to avoid lag.
95.         GameObject clone2 = Instantiate(respawnParticles, new
    Vector2(spawnPoint.transform.position.x, spawnPoint.transform.position.y + 2), new
    Quaternion(0f, 0f, 0f, 0f)) as GameObject; //Instantiate particles upon respawn.
96.         clone2.transform.rotation = Quaternion.Euler(270, 0, 0); //Ensure correct direction of
    particle system.
97.         Destroy(clone2, 2f); //Destroy the particle system to avoid lag.
98.         playerScript.gameObject.transform.localPosition = new
    Vector2(spawnPoint.transform.position.x, spawnPoint.transform.position.y + 3);
```

```
99.         playerScript.playerRgdBdy.velocity = new Vector2(0f, 2f); //Have the main character
    "pop out" of his fade-in, giving a slight, elevating jolt.
100.            if (!(SceneManager.GetActiveScene().name == "tfPalaceBoss" &&
    GameObject.FindWithTag("Enemy") == null)) //If the enemy is a boss, do not respawn.
101.            {
102.                StartCoroutine(ensureRespawn());
103.            }
104.            audioMast.PlaySound(playerScript.deathNoise);
105.        }
106.
107.        public void respawnEnemies() //Respawn the enemies after the player has died.
108.        {
109.            for (int i = 0; i < enemyGameObjects.Length; i++) //Activate each of the scene's
    enemies.
110.            {
111.                enemyGameObjects[i].SetActive(true); //Activate enemy, 'i.'
112.                enemyGameObjects[i].GetComponent<Enemy>().wBoxEnemy.SetActive(false);
    //Make sure the weapon colliders are inactive.
113.            }
114.        }
115.
116.        public IEnumerator ensureRespawn() //With regard to parries, if Unity fails to wait for a
    fraction of a second via this specific function, the enemy against whom the player parries fails
    to respawn.
117.        {
118.            yield return new WaitForSeconds(0.05f);
119.            respawnEnemies();
120.        }
121.    }
```

NextLevelManager.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.SceneManagement;
5.
6.  public class NextLevelManager : MonoBehaviour {
7.
8.      public Transform exitPoint;
9.
10.     public string levelToLoad;
11.
12.     private PlayerController playerScript;
13.
14.     private Animator levelFadeAnim;
15.
16.     private AudioMaster audioMast;
17.
18.     private NLGManager nlgMan;
19.
20.     // Use this for initialization
21.     void Start () {
22.         levelFadeAnim = GameObject.Find("LevelFade").GetComponent<Animator>();
23.         audioMast = AudioMaster.instance;
24.         exitPoint = GameObject.FindWithTag("Exit").GetComponent<Transform>();
25.         playerScript = GameObject.Find("Player").GetComponent<PlayerController>();
26.         nlgMan = playerScript.gameObject.GetComponent<NLGManager>();
27.     }
28.
```

```
29.    void Update () {
30.       nlgMan.distFromGoal =
Mathf.Sqrt(Mathf.Pow(playerScript.gameObject.transform.position.x -
31.                            gameObject.transform.position.x, 2f) +
32.                            Mathf.Pow(playerScript.gameObject.transform.position.y -
33.                            gameObject.transform.position.y, 2f)); //PYTHSWAGOREAN THEOREM
34.       if (exitPoint != null && playerScript.gameObject.transform.position.x <=
exitPoint.position.x + 3 &&
35.          playerScript.gameObject.transform.position.x >= exitPoint.position.x - 3 &&
36.          playerScript.gameObject.transform.position.y <= exitPoint.position.y + 3 &&
37.          playerScript.gameObject.transform.position.y >= exitPoint.position.y - 3) //If the
player has reached the exit, perform the following.
38.       {
39.          StartCoroutine(NextScene());
40.       }
41.    }
42.
43.    public IEnumerator NextScene()
44.    {
45.       nlgMan.deathsDuringLevel = 0;
46.       nlgMan.killsDuringLevel = 0;
47.       levelFadeAnim.SetBool("isOver", true); //Begin the screen-darkening animation.
48.       yield return new WaitForSeconds(0.12f); //Give the animation some time.
49.       audioMast.StopAll(); //Don't allow noises to bleed into the next scene.
50.       SceneManager.LoadScene(levelToLoad);
51.       levelFadeAnim.SetBool("isOver", false); //End the animation.
52.    }
53. }
```

LobbyController.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class LobbyController : MonoBehaviour {
6.
7.     private PlayerController playerScript;
8.
9.     private NextLevelManager nextLevelScript;
10.
11.    private Transform twiVal;
12.    private Transform shrWoods;
13.    private Transform nmPass;
14.
15.    [SerializeField]
16.    private GameObject gateClosed;
17.
18.    [SerializeField]
19.    private GameObject gateOpen;
20.
21.    [SerializeField]
22.    private GameObject finalAssistText;
23.
24.    void Start () {
25.       if (GameMaster.bossCount >= 4 && GameMaster.bossCount < 6) //If the first four
bosses have been killed, open the lobby's gate.
26.       {
27.          gateClosed.SetActive(false);
28.          gateOpen.SetActive(true);
```

```
        }
        playerScript = GameObject.Find("Player").GetComponent<PlayerController>();
        nextLevelScript =
GameObject.FindWithTag("Exit").GetComponent<NextLevelManager>();
        twiVal = GameObject.Find("mournerCircleExitTwiVal").GetComponent<Transform>();
        shrWoods =
GameObject.Find("mournerCircleExitShrWoods").GetComponent<Transform>();
        nmPass = GameObject.Find("mournerCircleExitnmPass").GetComponent<Transform>();
    }

    void Update () {
        if (GameMaster.bossCount >= 6) //If the player has completed the game, deactivate the
lobby's exit, and activate the final message.
        {
            finalAssistText.SetActive(true);
            nextLevelScript.exitPoint.gameObject.SetActive(false);
        }
        if (Input.GetKeyDown("s")) //When the player presses this key, perform the following.
        {
            if (GameMaster.mtwivalbool && playerScript.gameObject.transform.position.x <
twiVal.position.x + 4 && playerScript.gameObject.transform.position.x > twiVal.position.x - 4)
//If the player is near the leftmost door, perform the following.
            {
                nextLevelScript.levelToLoad = "twiValAlef"; //Have the next level be the first part of
Twilight Valley.
                GameMaster.mtwivalbool = false; //The player may not reenter the area.
                StartCoroutine(nextLevelScript.NextScene()); //Begin the next level.
            }
            else if (GameMaster.mshrwoodsbool && playerScript.gameObject.transform.position.x
< shrWoods.position.x + 4 && playerScript.gameObject.transform.position.x >
shrWoods.position.x - 4) //If the player is near the middle door, perform the following.
            {
                nextLevelScript.levelToLoad = "shrWoodsAlef"; //Have the next level be the first
part of Shrieking Woods.
                GameMaster.mshrwoodsbool = false; //The player may not reenter the area.
                StartCoroutine(nextLevelScript.NextScene()); //Begin the next level.
            }
            else if (GameMaster.mnmpassbool && playerScript.gameObject.transform.position.x
< nmPass.position.x + 4 && playerScript.gameObject.transform.position.x > nmPass.position.x
- 4) //If the player is near the rightmost door, perform the following.
            {
                nextLevelScript.levelToLoad = "nmPassAlef"; //Have the next level be the first part
of No Man's Pass.
                GameMaster.mnmpassbool = false; //The player may not reenter the area.
                StartCoroutine(nextLevelScript.NextScene()); //Begin the next level.
            }
        }
    }
}
```

GameMaster.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class GameMaster : MonoBehaviour {

```

```
9.    public static GameMaster gameMaster;
10.   private static string mLevelToSave;
11.   public static bool mtwivalbool;
12.   public static bool mshrwoodsbool;
13.   public static bool mnmpassbool;
14.   public static int bossCount;
15.
16.   public string mouseClick = "clickButton";
17.
18.   private AudioMaster audioMast;
19.
20.   void Awake()
21.   {
22.
23.      if (gameMaster == null)
24.      {
25.         DontDestroyOnLoad(gameObject);
26.         gameMaster = this;
27.      }
28.      else if (gameMaster != this)
29.      {
30.         Destroy(gameObject);
31.      }
32.   }
33.
34.   public void startGame()
35.   {
36.      AudioMaster.instance.PlaySound(mouseClick);
37.      mtwivalbool = true;
38.      mshrwoodsbool = true;
39.      mnmpassbool = true;
40.      bossCount = 0;
41.      SceneManager.LoadScene("tivAlef");
42.   }
43.
44.   public void continueGame()
45.   {
46.      AudioMaster.instance.PlaySound(mouseClick);
47.      Load();
48.      if (mLevelToSave != null)
49.      {
50.         SceneManager.LoadScene(mLevelToSave);
51.      }
52.   }
53.
54.   public void Save()
55.   {
56.      BinaryFormatter bFormatter = new BinaryFormatter();
57.      FileStream saveFile = File.Create(Application.persistentDataPath + "/playerinfo.dat");
58.      playerInfo info = new playerInfo();
59.      info.savedLevel = SceneManager.GetActiveScene().name;
60.      info.twivalbool = mtwivalbool;
61.      info.shrwoodsbool = mshrwoodsbool;
62.      info.nmpassbool = mnmpassbool;
63.      info.bosscountstored = bossCount;
64.      bFormatter.Serialize(saveFile, info);
65.      saveFile.Close();
66.   }
67.
68.   public void Load()
69.   {
```

```
70.         if(File.Exists(Application.persistentDataPath + "/playerinfo.dat"))
71.         {
72.             BinaryFormatter bFormatter = new BinaryFormatter();
73.             FileStream saveFile = File.Open(Application.persistentDataPath + "/playerinfo.dat",
    FileMode.Open);
74.             playerInfo info = (playerInfo) bFormatter.Deserialize(saveFile);
75.             saveFile.Close();
76.
77.             mLevelToSave = info.savedLevel;
78.             mtwivalbool = info.twivalbool;
79.             mshrwoodsbool = info.shrwoodsbool;
80.             mnmpassbool = info.nmpassbool;
81.             bossCount = info.bosscountstored;
82.         }
83.     }
84.
85.     public void quitGame()
86.     {
87.         AudioMaster.instance.PlaySound(mouseClick);
88.         Save();
89.         SceneManager.LoadScene("mainMenu");
90.     }
91.
92.     public void exitToDesktop()
93.     {
94.         AudioMaster.instance.PlaySound(mouseClick);
95.         Application.Quit();
96.     }
97.
98.     [Serializable]
99.     class playerInfo
100.         {
101.             public string savedLevel;
102.             public bool twivalbool;
103.             public bool shrwoodsbool;
104.             public bool nmpassbool;
105.             public int bosscountstored;
106.         }
107.     }
```

Noun.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class Noun
6.  {
7.
8.      private bool isProper; //This variable determines whether the noun is a proper noun.
9.      private string singular; //This variable is always the same as the name of the class
    instance.
10.     private string plural; //This variable is usually the same as the name of the class instance
    plus 's'.
11.     private bool beginsWithVowel;
12.
13.     public Noun(string _singular, bool _beginsWithVowel) //This constructor is for most
    nouns.
14.     {
15.
```

```
16.         isProper = false;
17.         singular = _singular;
18.         plural = _singular + "s";
19.         beginsWithVowel =_beginsWithVowel;
20.     }
21.
22.     public Noun(bool _isProper, string _singular, bool _beginsWithVowel)
23.     {
24.         isProper = _isProper;
25.         singular = _singular;
26.         plural = _singular + "s";
27.         beginsWithVowel = _beginsWithVowel;
28.     }
29.
30.     public Noun(bool _isProper, string _singular, string _plural, bool _beginsWithVowel)
    //This constructor is more complete than the previous one.
31.     {
32.         isProper = _isProper;
33.         singular = _singular;
34.         plural = _plural;
35.         beginsWithVowel = _beginsWithVowel;
36.         //usesArticle0 = _usesArticle0;
37.     }
38.
39.     public string nounToString()
40.     {
41.         return "isProper: " + isProper + " singular: " + singular + " plural: " + plural;
42.     }
43.
44.     public string getSingular()
45.     {
46.         return singular;
47.     }
48.
49.     public string getPlural()
50.     {
51.         return plural;
52.     }
53.
54.     public bool getIsProper()
55.     {
56.         return isProper;
57.     }
58.
59.     public bool getBeginsWithVowel()
60.     {
61.         return beginsWithVowel;
62.     }
63. }
```

Verb.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class Verb {
6.
7.      private string infinitive = "to verb";
8.      private string root = "verb";
```

```
9.      private string tpSingular = "it verbs";
10.     private string presentParticiple = "verbing";
11.     private string past = "verbed";
12.     private string pastParticiple = "verben";
13.
14.     public Verb(string _root)
15.     {
16.         infinitive = "to " + _root;
17.         root = _root;
18.         tpSingular = _root + "s";
19.         presentParticiple = _root + "ing";
20.         past = _root + "ed";
21.         pastParticiple = past;
22.     }
23.
24.     public Verb(string _root, string _past)
25.     {
26.         infinitive = "to " + _root;
27.         root = _root;
28.         tpSingular = _root + "s";
29.         presentParticiple = _root + "ing";
30.         past = _past;
31.         pastParticiple = _past;
32.     }
33.
34.     public Verb(string _root, string _presentParticiple, string _past)
35.     {
36.         infinitive = "to " + _root;
37.         root = _root;
38.         tpSingular = _root + "s";
39.         presentParticiple = _presentParticiple;
40.         past = _past;
41.         pastParticiple = _past;
42.     }
43.
44.     public Verb (string _root, string _presentParticiple, string _past, string _pastParticiple)
45.     {
46.         infinitive = "to " + _root;
47.         root = _root;
48.         tpSingular = _root + "s";
49.         presentParticiple = _presentParticiple;
50.         past = _past;
51.         pastParticiple = _pastParticiple;
52.     }
53.
54.     public Verb(string _infinitive, string _root, string _tpSingular, string _presentParticiple,
    string _past, string _pastParticiple)
55.     {
56.         infinitive = _infinitive;
57.         root = _root;
58.         tpSingular = _tpSingular;
59.         presentParticiple = _presentParticiple;
60.         past = _past;
61.         pastParticiple = _pastParticiple;
62.     }
63.
64.     public string verbToString()
65.     {
66.         return "infinitive: " + infinitive + " root: " + root + " tpSingular: " + tpSingular + "
    presentParticiple: " + presentParticiple + " past: " + past + " pastParticiple: " + pastParticiple;
67.     }
```

```
68.
69.     public string getInfinitive()
70.     {
71.         return infinitive;
72.     }
73.
74.     public string getRoot()
75.     {
76.         return root;
77.     }
78.
79.     public string getTpSingular()
80.     {
81.         return tpSingular;
82.     }
83.
84.     public string getPresentParticiple()
85.     {
86.         return presentParticiple;
87.     }
88.
89.     public string getPast()
90.     {
91.         return past;
92.     }
93.
94.     public string getPastParticiple()
95.     {
96.         return pastParticiple;
97.     }
98. }
```

NLGManager.cs

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class NLGManager : MonoBehaviour {
6.
7.      //I place both verbs and nouns in arrays so that I can access each object smoothly by way
    of tracking indices.
8.      //create nounarray outside of start(?)
9.
10. #pragma warning disable 0219 //NEVER FORGET ABOUT THIS
11.
12.     Noun killer = new Noun("killer", false), enemy = new Noun(false, "enemy", "enemies",
    true), //I can't have 'enemys.'
13.     exile = new Noun("exile", true), imp = new Noun("imp", true), knight = new
    Noun("knight", false), sword = new Noun("sword", false),
14.     axe = new Noun("axe", true), gap = new Noun("gap", false), drop = new Noun("drop",
    false), wall = new Noun("wall", false),
15.     platform = new Noun("platform", false), Disciple = new Noun(true, "Disciple", false),
    Nittonio = new Noun(true, "Nittonio", false),
16.     Sevryna = new Noun(true, "Sevryna", false), Panic = new Noun(true, "Panic", false),
    Qitma = new Noun(true, "Qitma", false),
17.     Qalem = new Noun(true, "Qalem", false), Kenzin = new Noun(true, "Kenzin", false),
    Infinite_Domain = new Noun(true, "Infinite Domain", true),
18.     Tear_in_the_Void = new Noun(true, "Tear in the Void", "Tears in the Void", false),
    Twilight_Valley = new Noun(true, "Twilight Valley", false),
```

```
19.     Rotwood_Keep = new Noun(true, "Rotwood Keep", false), Shrieking_Woods = new
    Noun(true, "Shrieking Wood", "Shrieking Woods", false),
20.     Halls_of_Panic = new Noun(true, "Hall of Panic", "Halls of Panic", false), No_Mans_Pass =
    new Noun(true, "No Man's Pass", "No Man's Passes", false),
21.     Tomb_of_the_Destroyer = new Noun(true, "Tomb of the Destroyer", "Tombs of the
    Destroyer", false), Kings_Path = new Noun(true, "King's Path", false),
22.     Faultless_Palace = new Noun(true, "Faultless Palace", false);
23.
24.     Verb kill = new Verb("kill"), die = new Verb("to die", "die", "dies", "dying", "died", "died"),
25.     fall = new Verb("fall", "fell"), jump = new Verb("jump"), leap = new Verb("leap"),
26.     fly = new Verb("to fly", "fly", "flies", "flying", "flew", "flown"),
27.     be = new Verb("to be", "am", "is", "are", "was", "been"), //Due to the verb's irregularity, I
    should consider the use of errant strings instead of Verb objects.
28.     respawn = new Verb("respawn"), stab = new Verb("stab", "stabbing", "stabbed"),
29.     run = new Verb("run", "running", "ran", "run"),
30.     obliterate = new Verb("obliterate", "obliterating", "obliterated"),
31.     smash = new Verb("to smash", "smash", "smashes", "smashing", "smashed", "smashed"),
32.     ricochet = new Verb("ricochet"), recoil = new Verb("recoil"),
33.     push = new Verb("to push", "push", "pushes", "pushing", "pushed", "pushed"),
34.     pull = new Verb("pull"), force = new Verb("force", "forcing", "forced"),
35.     stop = new Verb("stop", "stopping", "stopped"), avoid = new Verb("avoid"),
36.     bolt = new Verb("bolt"), move = new Verb("move", "moving", "moved");
37.
38.     public bool hasFallen = false, hasBeenStabbed = false, hasStabbed = false,
    quickSuccessionFalls = false,
39.         quickSuccessionStabs = false, hasKilledBoss = false, boltMeterEmpty = false,
    boltUnusedForLong = false,
40.         aerialKill = false;
41.
42.     public int deathsDuringLevel = 0, deathsDuringGame = 0,  killsDuringLevel = 0,
    killsDuringGame = 0, inAir = 0;
43.
44.     private int last = 0;
45.
46.     public float distFromGoal = 0;
47.
48.     public string[] mildlySadWords = { "out-of-sorts", "unfortunate", "regrettable", "pitiful",
    "pitiable" };
49.
50.     public string[] sadWords = { "blah", "lamentable", "disappointing", "lame", "dismal" };
51.
52.     public string[] verySadWords = { "disgusting", "reprehensible", "disgraceful", "miserable",
    "traumatic" };
53.
54.     public string[] theAdverbs = { "truly", "ostensibly", "harshly", "grotesquely" };
55.
56.     public string[] subjectPronouns = {  "he", "she", "who", "whoever", "you", "I", "we",
    "they" };
57.
58.     public string[] objectPronouns = {  "him", "her", "whom", "whomever", "you", "me", "us",
    "them" };
59.
60.     public string[] prepositions = { "with", "in", "at", "by", "before", "between", "from", "on",
    "over", "to" };
61.
62.     public string[] conjunctions = { "for", "nor", "or", "and", "but", "yet", "so" };
63.
64.     public string[] articles = { "an", "a", "the", "that", "this", "every", "each", "some", "most",
    "these", "those"};
65.
66.     public string[] whWords = { "who", "what", "when", "where", "why", "how" };
```

```
67.
68.     public string S = "";
69.
70.     void Update () {
71.         if ((aerialKill || hasFallen || hasStabbed || hasBeenStabbed || boltMeterEmpty ||
    quickSuccessionFalls || quickSuccessionStabs || hasKilledBoss))
72.         {
73.             ruleSwitch();
74.         }
75.     }
76.
77.     void ruleSwitch()
78.     {
79.         Noun[] nounArray = { killer, enemy, exile, imp, knight, sword, axe, drop, gap, wall,
    platform, Disciple,
80.         Nittonio, Sevryna, Panic, Qitma, Qalem, Kenzin, Twilight_Valley, Rotwood_Keep,
    No_Mans_Pass, Infinite_Domain, Tear_in_the_Void, Shrieking_Woods, Halls_of_Panic,
    Tomb_of_the_Destroyer,
81.         Kings_Path, Faultless_Palace };
82.
83.         //12 through 20 can't use "the"
84.         Verb[] verbArray = { kill, die, fall, jump, leap, fly, respawn, run, stab, obliterate, smash,
    ricochet, recoil, push, pull, force, stop, be, avoid, bolt, move };
85.
86.         //Grammar Rules... Still working on precedence and creating syntactical conditionals,
    most of which will be random...
87.         string NP = "", NPObj = "", VP = "", AdjectiveVar = sadWords[Random.Range(0,4)],
    AdverbVar = theAdverbs[Random.Range(0,3)], ConjunctionVar =
    conjunctions[Random.Range(0,6)], PrepositionVar = prepositions[Random.Range(0,9)],
    ArticleVar = articles[Random.Range(0,8)], sPronounVar =
    subjectPronouns[Random.Range(0,7)], oPronounVar = objectPronouns[Random.Range(0,7)],
    whWord = whWords[Random.Range(0,5)];
88.         int temp = Random.Range(0, 27); //Choose among the nouns (for the subject).
89.         int tempObj = Random.Range(0, 27); //Choose among the nouns (for the object).
90.         int select = Random.Range(0, 11);
91.         int singularOrPlural = Random.Range(0, 1); //Choose whether the subject will be
    singular or plural.
92.         int singularOrPluralObj = Random.Range(0, 1); //Choose whether the object will be
    singular or plural.
93.         int tenseSwitch = Random.Range(0, 4); //Choose among the existing tenses.
94.         int useGeneration = Random.Range(0, 2);
95.         if (select == last)
96.         {
97.             ruleSwitch();
98.         }
99.         if (useGeneration == 0 || boltMeterEmpty)
100.            {
101.                if (quickSuccessionFalls)
102.                {
103.                    if (boltUnusedForLong)
104.                    {
105.                        string[] sentences = { "death comes for all who fail to use bolt.", "you have
    abilities.  you should consider using them.", "you don't seem to be understanding this whole
    bolt thing.", "the bolt ability seems not to be getting through this one's mind", "avert death
    with dexterity.", "dexterity might make your lives a little easier.", "you could try bolting.  you
    can't possibly die more often.", "bolting, shmolting.  though it could seriously help you,
    genius.", "you could avoid death with some speed.", "use bolt, every once in a while.", "a
    running start wouldn't kill you.", "move. fast. then. jump. good lord." };
106.                        Debug.LogWarning("NON-GENERATED: " + sentences[select]);
107.                        S = sentences[select];
108.                        last = select;
```

```
109.                quickSuccessionFalls = false;
110.                hasFallen = false;
111.                boltUnusedForLong = false;
112.            }
113.            else if (inAir > 55)
114.            {
115.                string[] sentences = { "'Look at me.  I'm the Disciple, and I'm the best person here at killing myself.'", "Not only are you getting sweet air, but you're getting the sweet release of death, too.", "It's not the fall that bores you.  It's the impact.", "Don't you become tired of that wretched noise?", "Get a life.", "Masochism becomes monotony.", "You're not going to move the planet.", "You can try to move the planet.  It's legal.", "You are as graceful as a meteor piloting a collapsing building.", "I feel worse for the floor than I do for you.", "You've shown the ground who's boss.", "Praise the ground." };
116.                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
117.                S = sentences[select];
118.                last = select;
119.                quickSuccessionFalls = false;
120.                hasFallen = false;
121.            }
122.            else if (distFromGoal > 125)
123.            {
124.                string[] sentences = { "somebody's frustrated.", "i fear you fail to see the point.", "This is not much of a plan.", "Consider jumping.  Try it.", "You know, these surfaces aren't that slippery.", "Have you any hobbies?", "Do you know what a hobby is?", "You are truly amoebic.", "You must be AFK.", "That's beyond bad.", "You're extremely coordinated.", "Crazy or stupid?" };
125.                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
126.                S = sentences[select];
127.                last = select;
128.                quickSuccessionFalls = false;
129.                hasFallen = false;
130.            }
131.            else if (distFromGoal > 60 && distFromGoal < 125)
132.            {
133.                string[] sentences = { "i don't think you're lazy.  i know you're nuts.", "certainly, you have a motive.", "I'm sure you have an excellent reason for this.", "You have a good reason to do this. I'm sure.", "Perhaps you are not as amoebic as I previously thought.", "You're putting in a strange amount of effort in the realm of failing.", "I don't understand.", "Just enough effort to kill yourself.  Fascinating.", "How incompetent can you possibly be?", "Are you, like, two?", "Unbelievable, you clown.", "Oh, wow." };
134.                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
135.                S = sentences[select];
136.                last = select;
137.                quickSuccessionFalls = false;
138.                hasFallen = false;
139.            }
140.            else
141.            {
142.                string[] sentences = { "you are adept at falling.", "falling into oblivion constantly is no way to go through life.", "Enough with the falling.", "Do something beside falling.  Just a suggestion.", "Keep typing on the keyboard, and you'll drive yourself crazy.", "You'd be making a lot more headway by doing nothing.", "Doing nothing would get you just as far.", "Stop touching stuff.", "Stop touching the keyboard for a moment.", "Calm down.", "There's more to this game than falling down and dying.", "Avoiding death might be worth your while." };
143.                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
144.                S = sentences[select];
145.                last = select;
146.                quickSuccessionFalls = false;
147.                hasFallen = false;
148.            }
```

```
149.            }
150.
151.            else if (hasFallen)
152.            {
153.                //Debug.LogWarning(distFromGoal);
154.                if (boltUnusedForLong)
155.                {
156.                    string[] sentences = { "learn how to bolt.", "it's time to use bolt.", "Think about using bolt at least once.", "Bolt might help you once or twice.", "Bolting is helpful.  I swear.", "I promise bolting will help you in the long run.  No pun intended.", "Hit 'shift' every now and then.", "Come on.  Use shift.", "Avoid death with some speed.", "Bolt, every once in a while.", "A running start wouldn't kill you.", "Move. Fast. Then. Jump. Good lord." };
157.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
158.                    S = sentences[select];
159.                    last = select;
160.                    hasFallen = false;
161.                    boltUnusedForLong = false;
162.                }
163.                else if (deathsDuringLevel % 60 == 0 && deathsDuringLevel >= 60)
164.                {
165.                    string[] sentences = { "you're becoming quite the base jumper.", "death comes for us all.  death comes for us a lot.", "Falling is overrated.", "If only I had a nickel for every time you did this.", "You're loving this.", "No one should want to do this.", "Maybe it's time to go outside and give your nerves a break.", "All precipitation and no success makes the Disciple a monotonous boy.", "You seem to be having a bit of trouble.  Maybe you should stop having trouble.", "This is easy.  That's why it's for immortal deities who're unable to die fully.", "All you need to do is fly across the map like a pinball.", "#JumpHigherDieLess" };
166.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
167.                    S = sentences[select];
168.                    last = select;
169.                    hasFallen = false;
170.                }
171.                else if (deathsDuringLevel % 10 == 0 && deathsDuringLevel >= 10)
172.                {
173.                    string[] sentences = { "ah, the monotony of death.", "you lose some, you lose some.", "Trying again is always an option.", "A winning move would be to avoid gameplay.", "By all means, continue.", "There's more death on its way, unfortunately.", "The futility of existence pesters us all.", "There's another ten.  Having problems with your footing?", "I'm sure you feel you've died enough.", "A death a minute keeps the sanity away.", "Who needs life when you have the ground?", "Acceptance or self-hatred?" };
174.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
175.                    S = sentences[select];
176.                    last = select;
177.                    hasFallen = false;
178.                }
179.                else if (inAir > 55)
180.                {
181.                    string[] sentences = { "tubular.", "what a neat aerial assassination on yourself.", "Banzaiiiii...", "Chill, Bill Murray in Groundhog Day.", "I grant you 10,000 points for that jump.  Points are worthless.", "You're not going to bounce.", "Be nice to your skeleton.", "You broke the fall with yourself.", "You are as graceful as a meteor.", "I feel worse for the floor than I do for you.", "You've shown the ground who's boss.", "Praise the ground." };
182.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
183.                    S = sentences[select];
184.                    last = select;
185.                    hasFallen = false;
186.                }
187.                else if (distFromGoal > 125)
188.                {
189.                    string[] sentences = { "what a great beginning.", "you're off to a good start.", "wonderful.", "At least fall later.", "There are other platforms to miss.", "You needn't
```

```
miss the first few platforms, honestly.", "If you can't deal with the drops, you can't deal with the bad guys.", "Deal with the drops to deal with the bad guys.", "Come on.", "You didn't even try.", "You're really coordinated.", "How did you turn on the computer you're using?" };
                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                    S = sentences[select];
                    last = select;
                    hasFallen = false;
                }
                else if (distFromGoal > 60 && distFromGoal < 125)
                {
                    string[] sentences = { "you could be doing worse.", "at least you've experienced the level.", "This is a stupid level, anyway.", "Oh, what difference does it make?", "Another death for the collection.", "That'll leave several marks, including emotional ones.", "You'll be fine, even if that happens a few more times.", "Okay, so, you died.  We've all been there.", "You were doing kind of well.", "Death annoys you, I'm sure.  It annoys me, too.", "Decent progress.", "Nope." };
                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                    S = sentences[select];
                    last = select;
                    hasFallen = false;
                }
                else if (distFromGoal < 60)
                {
                    string[] sentences = { "brutal.", "gravity's not a nice person.", "What a lamentable trajectory.", "You were off by a little bit.", "Woosh.", "Lives could be worse.", "You've died so close to the end.  Neat.", "We were overdue for a death.", "Wow.", "You must have done something very bad to deserve this.", "This is what being immediately above par feels like.", "You embarked on quite a trek, before death.  Now, you can do it, again." };
                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                    S = sentences[select];
                    last = select;
                    hasFallen = false;
                }
            }

            else if (quickSuccessionStabs)
            {
                if (boltUnusedForLong)
                {
                    string[] sentences = { "escapism works.", "fear not.  you can bolt.", "Getting stabbed is intensely overrated.", "From bolt's absence comes cuts and bruises.  Who knew?", "'Pie jesu domine,' *shank* 'Dona peis requiem,' *shank*", "Slowpoke.", "If you get stabbed enough, you pretty much just become a donut or a bagel.  No bolting, but rolling, which works as well.", "If only you bolted as often as they stabbed you.", "Bolt past them.", "Use bolt to dodge their blows.", "Stabbing a moving target is harder.", "Concept: Move fast to kill fast." };
                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                    S = sentences[select];
                    last = select;
                    quickSuccessionStabs = false;
                    hasBeenStabbed = false;
                    boltUnusedForLong = false;
                }
                else if (distFromGoal > 125)
                {
                    string[] sentences = { "the AOL guy says, 'You got stabbed.'", "in the beginning, there were knives.", "Access intensely denied.", "These guys could have given you a little more time.", "Already, the stabbing begins.", "They should be stabbing things their own size.", "An immediate debacle.", "Get stabbed fast to succeed.", "Surprise swords.", "Swords don't let you come very close.", "Cowabunga.", "Do you even know how to breathe?" };
                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
```

```
229.                    S = sentences[select];
230.                    last = select;
231.                    quickSuccessionStabs = false;
232.                    hasBeenStabbed = false;
233.                }
234.                else if (distFromGoal > 60 && distFromGoal < 125)
235.                {
236.                    string[] sentences = { "i appreciate your ability to maintain your current rate of being stabbed.", "Practice makes Polonius.", "At least you could be getting stabbed more often.", "Mr. Hyuga delivers his otherwise quick blows at a slower pace.", "The stabbings are spaced apart so well.  It's like a soap opera.", "If you're offering Morse Code signals, what's a dot, and what's a dash?", "AS entertaining as these murders are, they seem ineffective.", "Something's not working.", "You were doing kind of well for a guy who likes to be stabbed.", "Those abysmal swords... always stabbing things.", "Possibly decent progress.", "The swords decline." };
237.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
238.                    S = sentences[select];
239.                    last = select;
240.                    quickSuccessionStabs = false;
241.                    hasBeenStabbed = false;
242.                }
243.                else
244.                {
245.                    string[] sentences = { "there's never a dull moment in the Infinite Domain.", "Way to dull some blades.", "You were just distracting them.  Of course!", "Feel free to stop getting stabbed.", "I dare you not to be stabbed, again.", "Do what you want.", "Getting stabbed isn't that interesting.", "Being stabbed is not the entire point of the game.", "Stabpocalypse.", "Come on.  Stop getting stabbed.", "Do you want this to happen to you?", "What gives with all the knives?" };
246.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
247.                    S = sentences[select];
248.                    last = select;
249.                    quickSuccessionStabs = false;
250.                    hasBeenStabbed = false;
251.                }
252.            }
253.
254.            else if (hasBeenStabbed)
255.            {
256.                if (boltUnusedForLong)
257.                {
258.                    string[] sentences = { "bolt might be of assistance.", "Don't get stabbed. Use bolt, instead.", "Go a little faster.  Make your life easier.", "Well, you haven't used bolt in a while.", "You can press the 'shift' key to go faster.", "You could have avoided that.", "Go.  Don't loiter.", "Standing still is punishable by death.", "Bolt past them.", "Use bolt to dodge their blows.", "Stabbing a moving target is harder.", "Concept: Move fast to kill fast." };
259.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
260.                    S = sentences[select];
261.                    last = select;
262.                    hasBeenStabbed = false;
263.                    boltUnusedForLong = false;
264.                }
265.                else if (deathsDuringLevel % 60 == 0 && deathsDuringLevel >= 60)
266.                {
267.                    string[] sentences = { "knives yield monotony.", "Knives can be upsetting, can't they?", "Julius Caesar?  Is that you?", "So many stabs.  So little diversity.", "That's a lot of murder.", "What difference does another death make?", "How many stabs could a Disciple receive if a Disciple could occasionally avoid stabs?", "A few evasive maneuvers wouldn't hurt.", "The abundance of weapons isn't making things easier for you.", "Dying is tough, especially when you get stabbed.", "Surely, you're used to this, by now.", "I'm not sure I understand your deep infatuation with death." };
```

```
268.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
269.                    S = sentences[select];
270.                    last = select;
271.                    hasBeenStabbed = false;
272.                }
273.                else if (deathsDuringLevel % 10 == 0 && deathsDuringLevel >= 10)
274.                {
275.                    string[] sentences = { "i suppose there could be more stabs.", "It's okay.  I
    don't even have a body worth stabbing.", "Maybe you really are a swarma, after all.", "You're
    still a pretty elusive swarma, at least.", "You've been stabbed.  Happens to the best of us.",
    "It's time for a change of behavior.", "Maybe you should change your behavior.", "Recall
    Einstein's definition of insanity.", "What a way to go.", "A death a minute keeps the sanity
    away.", "Who needs life when you have steel?", "Hooray for knives." };
276.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
277.                    S = sentences[select];
278.                    last = select;
279.                    hasBeenStabbed = false;
280.                }
281.                else if (inAir > 55)
282.                {
283.                    string[] sentences = { "an interceptor.", "good job, bad guy.", "I'm
    surprised one of them was able to do that.", "Maybe these enemies aren't so dumb.", "I could
    have sworn these guys were dumber.", "They shouldn't have managed that.", "I can't believe
    they did that.", "An enemy with that level of precision is not okay.", "Interception.", "They've
    saved you from the clutches of the ground.", "Sweet air makes for catastrophic collisions.
    Tradeoffs.", "Aerial murder, shmaerial murder." };
284.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
285.                    S = sentences[select];
286.                    last = select;
287.                    hasBeenStabbed = false;
288.                }
289.                else if (distFromGoal > 125)
290.                {
291.                    string[] sentences = { "Well, then...", "The enemies' minds are made up.",
    "The level ends as the level starts: abruptly, and without cutscenes.", "That's too bad.",
    "Shame.", "Doing great, already.", "Would you look at that?", "I'm sad and surprised,
    actually.", "Surprise swords.", "Swords don't let you come very close.", "Cowabunga.", "Do you
    even know how to breathe?" };
292.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
293.                    S = sentences[select];
294.                    last = select;
295.                    hasBeenStabbed = false;
296.                }
297.                else if (distFromGoal > 60 && distFromGoal < 125)
298.                {
299.                    string[] sentences = { "Not the best time to be stabbed.", "There are better
    times at which to be stabbed.", "Looks like it's going to be one of those days.", "True ambition
    is rebounding from impertinent labor.", "Being stabbed halfway through a level builds
    character.", "Sometimes, the most ambitious person must settle for character building.",
    "Perhaps you've died with style.", "I guess you could be dying a little more often.", "You were
    doing kind of well for a guy who likes to be stabbed.", "Those abysmal swords... always
    stabbing things.", "Possibly decent progress.", "The swords decline." };
300.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
301.                    S = sentences[select];
302.                    last = select;
303.                    hasBeenStabbed = false;
304.                }
305.                else if (distFromGoal < 60)
306.                {
307.                    string[] sentences = { "Yahtzee.", "So close, yet so skewered.", "Looking
    like a turnstile at the end of the level.  Sheesh.", "Yeah, that's not good.", "That's a far cry
```

```
from good.", "That's the opposite of good.", "Wouldn't it have been nice to be stabbed at any other time?", "Wrong place, wrong time.  Enemies are mean.", "My word.", "Venture to the pentagram.  Stray from enemies.", "Your ultimate success would be incomplete without another death.", "What a buzzkill." };
                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                S = sentences[select];
                last = select;
                hasBeenStabbed = false;
            }
        }

        else if (hasKilledBoss)
        {
            if (boltUnusedForLong)
            {
                string[] sentences = { "Bolt might've helped you, but you seem to have prevailed, anyway.", "Bolt might've made things easier.", "Giving yourself more of a challenge, by abstaining from bolt, I see.", "Oh you could have done that a little faster.", "Don't forget that bolt helps you kill stuff faster.", "You could have killed the boss faster with bolt.", "You could have spiced up that kill with a bolt.", "Aw. You didn't bolt.", "Without help from bolt.", "You didn't need very much help with that at all.", "That's a pretty legit strike from the heavens and whatnot, and you didn't have to use bolt.", "You're making some kind of point, perhaps." };
                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                S = sentences[select];
                last = select;
                hasKilledBoss = false;
                hasStabbed = false;
                boltUnusedForLong = false;
            }
            else if (aerialKill)
            {
                string[] sentences = { "Acrobatics!", "Killing bosses is such good exercise.", "A boss-defying leap.", "What a spectacular way to kill something.", "That was pretty cool.", "Show off.", "You're just showing off.", "There you go.", "Now, you're the boss.", "Elegant.", "That's a pretty legit strike from the heavens and whatnot.", "A spectacle." };
                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                S = sentences[select];
                last = select;
                hasKilledBoss = false;
                hasStabbed = false;
            }
        }

        else if (hasStabbed)
        {
            if (boltUnusedForLong)
            {
                string[] sentences = { "You could kill even more enemies if you used bolt.", "Bolt might yield additional kills.", "You could kill more enemies with bolt.", "Bolt could help you kill a ton of enemies.", "Killing enemies might be even easier for you if you throw in a bolt every now and then.", "Bolt would help you kill the bad guys.", "Swords are cool, but bolt makes the bad guys go away, too.", "Use bolt.  Kill faster.", "And without using bolt for a while. Impressive.", "Look, ma. No bolting.", "If you used bolt, you'd actually be doing better.", "Using bolt might allow you to kill more." };
                //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
                S = sentences[select];
                last = select;
                hasStabbed = false;
            }
            else if (deathsDuringLevel % 10 == 0 && deathsDuringLevel >= 10)
```

```
349.                {
350.                    string[] sentences = { "Look who's making a comeback.", "Now, you're angry.", "That's it. Become a menace.", "Teach those ones and zeroes who's boss.", "Disciple smash!", "Give them the old two piece and a biscuit.", "How the turns have tabled.", "How the tables have turned.", "You're making a comeback.", "Keep at the killing.", "A quality kill.", "Well, look at that." };
351.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
352.                    S = sentences[select];
353.                    last = select;
354.                    hasStabbed = false;
355.                }
356.                else
357.                {
358.                    string[] sentences = { "Tango down.", "That was pretty nice.", "Word.", "Hello, sword.", "Eat swords, filth.", "Solid swing.", "Bonk.", "Hya.", "Et tu?", "Well, aren't you a foolish samurai warrior wielding a magic sword?", "I see, you like a good kebab.", "Like a glove." };
359.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
360.                    S = sentences[select];
361.                    last = select;
362.                    hasStabbed = false;
363.                }
364.            }
365.
366.            else if (boltMeterEmpty)
367.            {
368.                if (quickSuccessionFalls)
369.                {
370.                    string[] sentences = { "Slow down.", "There's such a thing as using bolt too often.", "Use bolt, but only for a limited time.", "Bolting helps the patient.", "Patience is necessary for proper bolting.", "Energy depletes quickly.", "Don't do everything too fast.", "Give 'shift' a break.", "You're falling like crazy.  Bolt could assist you.", "Stop bolting. Allow for a recharge.", "Don't tire yourself out.", "You can't run like that, forever." };
371.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
372.                    S = sentences[select];
373.                    last = select;
374.                    boltMeterEmpty = false;
375.                }
376.                else if (hasBeenStabbed)
377.                {
378.                    string[] sentences = { "Give yourself enough energy to bolt out of there.", "Make sure you have enough energy for an escape.", "Escaping is a lot harder without bolt.", "Not enough 'juice.'", "Give bolt some time to charge before you need it.", "Try bolting out of that one, again.", "Be careful about your energy.", "Mind your energy.", "You haven't enough 'juice,' as they say.", "Stop bolting. Allow for a recharge.", "Don't tire yourself out.", "You can't run like that, forever." };
379.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
380.                    S = sentences[select];
381.                    last = select;
382.                    boltMeterEmpty = false;
383.                }
384.                else if (hasStabbed)
385.                {
386.                    string[] sentences = { "What a swift kill.", "A masterful use of the bolt ability.", "Nice bolt.", "A brilliant thrust.", "That's a good shot.", "Quality hit.", "Good one!", "You got 'em!", "You seem to prevail without using bolt.", "Stop bolting. Allow for a recharge.", "Don't tire yourself out.", "You can't run like that, forever." };
387.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
388.                    S = sentences[select];
389.                    last = select;
390.                    boltMeterEmpty = false;
```

```
391.                }
392.                else if (quickSuccessionStabs)
393.                {
394.                    string[] sentences = { "you'd be stabbed a bit less if you used bolt
    properly.", "stop bolting. Allow for a recharge.", "Don't tire yourself out.", "You can't run like
    that, forever." };
395.                    //Debug.LogWarning("NON-GENERATED: " + sentences[select]);
396.                    S = sentences[select];
397.                    last = select;
398.                    boltMeterEmpty = false;
399.                }
400.                else
401.                {
402.                    boltMeterEmpty = false;
403.                }
404.            }
405.        }
406.        else {
407.            if (hasFallen)
408.            {
409.                temp = Random.Range(7, 8);
410.                tempObj = 11;
411.                tenseSwitch = Random.Range(0, 3);
412.            }
413.            else if (hasBeenStabbed)
414.            {
415.                temp = Random.Range(0, 1);
416.                tempObj = 11;
417.                tenseSwitch = Random.Range(0, 3);
418.            }
419.            else if (hasStabbed)
420.            {
421.                temp = 11;
422.                tempObj = Random.Range(0, 1);
423.                tenseSwitch = Random.Range(0, 3);
424.            }
425.            else
426.            {
427.                return;
428.            }
429.            int articleSwitch = Random.Range(0, 1); //Choose whether to use certain articles
    or not.
430.            int articleSwitchObj = Random.Range(0, 1); //Choose whether to use certain
    articles with regard to the object.
431.            int usePronoun = Random.Range(0, 1); //Determine whether to have a pronoun
    be the subject.
432.            int usePronounObj = Random.Range(0, 1); //Determine whether to have a
    pronoun be the object.
433.
434.            string NounVar;
435.
436.            if (singularOrPlural == 0 && (temp <= 12 || temp >= 20))
437.            {
438.                NounVar = nounArray[temp].getSingular();
439.            }
440.
441.            else
442.            {
443.                NounVar = nounArray[temp].getPlural();
444.            }
445.
```

```
446.            string Object;
447.
448.            if (singularOrPluralObj == 0 && (tempObj <= 12 || tempObj >= 20))
449.            {
450.               Object = nounArray[tempObj].getSingular();
451.               if (hasFallen || hasBeenStabbed || hasStabbed)
452.               {
453.                  Object = nounArray[tempObj].getSingular();
454.               }
455.            }
456.
457.            else
458.            {
459.               Object = nounArray[tempObj].getPlural();
460.            }
461.
462.            string VerbVar = "";
463.            int verbChoice = 0;
464.            if (hasFallen)
465.            {
466.               verbChoice = 0;
467.            }
468.            else if (hasBeenStabbed || hasStabbed)
469.            {
470.               verbChoice = Random.Range(8, 10);
471.            }
472.            switch (tenseSwitch)
473.            {
474.               case 0:
475.                  VerbVar = verbArray[verbChoice].getPast();
476.                  break;
477.
478.               case 1:
479.                  if (singularOrPlural == 0)
480.                  {
481.                     VerbVar = verbArray[verbChoice].getTpSingular();
482.                  }
483.                  else
484.                  {
485.                     VerbVar = verbArray[verbChoice].getRoot();
486.                  }
487.                  break;
488.
489.
490.
491.               case 2:
492.                  if (singularOrPlural == 0 && (temp <= 12 || temp >= 20))
493.                  {
494.                     VerbVar = "has " + verbArray[verbChoice].getPastParticiple();
495.                  }
496.                  else
497.                  {
498.                     VerbVar = "have " + verbArray[verbChoice].getPastParticiple();
499.                  }
500.                  break;
501.
502.               case 3:
503.                  VerbVar = "had " + verbArray[Random.Range(0, 16)].getPastParticiple();
504.                  break;
505.
506.               case 4:
```

```
507.                VerbVar = "will " + verbArray[Random.Range(0, 16)].getRoot();
508.                break;
509.            }
510.
511.            VP = VerbVar;
512.
513.            if (nounArray[temp].getIsProper())
514.            {
515.                if (temp >= 12 && temp <= 20)
516.                {
517.                    NP = NounVar;
518.                }
519.
520.                else
521.                {
522.                    NP = articles[2] + " " + NounVar;
523.                }
524.            }
525.
526.            else
527.            {
528.                if (articleSwitch == 1)
529.                {
530.                    NP = articles[Random.Range(2, 6)] + " " + NounVar;
531.                }
532.
533.                if (singularOrPlural == 1)
534.                {
535.                    NP = articles[Random.Range(7, 10)] + " " + NounVar;
536.                }
537.
538.                else
539.                {
540.                    if (nounArray[temp].getBeginsWithVowel())
541.                    {
542.                        NP = articles[0] + " " + NounVar;
543.                    }
544.
545.                    else
546.                    {
547.                        NP = articles[1] + " " + NounVar;
548.                    }
549.
550.                    if (usePronoun == 1)
551.                    {
552.                        NP = subjectPronouns[Random.Range(0, 3)];
553.                        if (singularOrPlural == 1)
554.                        {
555.                            NP = subjectPronouns[Random.Range(4, 7)];
556.                        }
557.                    }
558.                }
559.            }
560.
561.            if (nounArray[tempObj].getIsProper())
562.            {
563.                if (tempObj >= 12 && tempObj <= 20)
564.                {
565.                    NPObj = Object;
566.                }
567.
```

```
568.             else
569.             {
570.                 NPObj = articles[2] + " " + Object;
571.             }
572.         }
573.
574.         else
575.         {
576.             if (articleSwitchObj == 1)
577.             {
578.                 NPObj = articles[Random.Range(2, 6)] + " " + Object;
579.             }
580.
581.             if (singularOrPluralObj == 1)
582.             {
583.                 NPObj = articles[Random.Range(7, 10)] + " " + Object;
584.             }
585.
586.             else
587.             {
588.                 if (nounArray[tempObj].getBeginsWithVowel())
589.                 {
590.                     NPObj = articles[0] + " " + Object;
591.                 }
592.
593.                 else
594.                 {
595.                     NPObj = articles[1] + " " + Object;
596.                 }
597.
598.                 if (usePronounObj == 1)
599.                 {
600.                     NPObj = objectPronouns[Random.Range(0, 7)];
601.                 }
602.             }
603.         }
604.
605.         hasFallen = false;
606.         quickSuccessionFalls = false;
607.         hasBeenStabbed = false;
608.         quickSuccessionStabs = false;
609.         boltMeterEmpty = false;
610.         hasKilledBoss = false;
611.         boltUnusedForLong = false;
612.         aerialKill = false;
613.         hasStabbed = false;
614.
615.         S = NP + " " + VP + " " + NPObj + ".";
616.         //Debug.LogWarning("articleSwitch: " + articleSwitch + " Sentence: " + S);
617.     }
618. }
619. }
```