GPT for the use of a Realistic Text-Adventure Game

Brent Goldstein

Troy, New York

goldsb2@rpi.edu

Motivation and Introduction

Salen and Zimmerman in their book Rules of Play: Game Design Fundamentals define games as "A system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome." [Salen, Zimmerman, 2010] This means that in a game, there are rules, actions, and consequences. For example, when you drop an item it falls to the floor due to gravity. The rule is that gravity pulls objects downwards, the action is dropping the item, and the consequence is the object falls to the floor. In a game, gravity is not a given, it is a rule that must be implemented. So then what might a game be like if there were a rule for every possible interaction?

If it were the case where every possible action the player can do has a quantifiable outcome, then that would make a video game comparable to reality. Many games and technologies, for example, augmented reality and virtual reality, attempt to create this sensation. There have been many strides in these fields, such as how they increase immersion, but whether they succeed in creating a reality-like game is still under research and review. [Fan, Liu, 2010]

Background

To make a game as complex as reality, the amount of rules and interactions that would need to be implemented is currently unfeasible. No game gives the player the freedom to do possibly anything they imagine. Many games attempt to mimic the complexities of reality but are still missing many interactions. For example, the game The Legend of Zelda: Breath of the Wild hosts many different programmed interactions. The player can explore the vast lands of a fantastical world with complex physics. The physics mimics reality, with objects that fall, roll, and bounce as one would expect. Even basic things like grass have a lot of interactions, it can blow in the wind or catch fire and spread if lit ablaze. Additionally, players can run around, fight enemies, and interact with non-playable characters (NPCs)

Actions in the game, however, are limited by its rules. Objects move with realistic physics, but the game's speedrunning community has found a multitude of ways of breaking Newton's laws in the game's engine. Grass interacts with wind and fire, but if the player wished to pick up some grass and eat it, or even just hold it in their hand, there are no possible ways of doing this or any other actions that are not explicitly built-in. Players cannot choose to frolic instead of run, fight non-enemies, or ask an NPC whatever line of dialogue they please. There must be a rule in place, and without any rule, there is no quantifiable outcome or consequence.

The main motivation for this project is to attempt to make a game that can expand its rules so that every possible action has a reaction. Normally, in a text-adventure game, if the user inputs a word the game does not understand, it will mark the input as invalid. In this project, even if the action is not programmed into the game it should still have a relevant output and a consequence from that action. This way, everything the player does is accounted for, if the player so chooses they can crawl instead of walk, eat grass, ask NPCs about how their day was, or any other action.

Related work

The most popular work to make use of GPT in forming a game that is like reality is AI Dungeon and AI Dungeon 2. It's a game where the story responds to the input of the player. It uses OpenAi and GPT-2 in order to adapt, respond, and output to any line of text the player feeds it. What GPT does is it attempts to predict the next text in a sequence. [Harris 2020] If the input was "I ate" it may fill that blank with "an apple" or any other line of text that could feasibly fit the end of that sentence. This is upscaled to fit larger text than just a simple sentence. So if the input is "Give me a story about eating an apple" the output will be a detailed retelling of someone eating an apple.

Ai Dungeone utilizes natural language processing or NLP. NLP uses machine learning to break down text in order to understand the meaning of the user input. [IBM] This way, virtually anything you input into Ai Dungeon and other similar programs is understood. Where Ai Dungeon and this work differ, however, is with a process called fine-tuning. Fine-tuning filters the results of the NLP and text completion so that the results fit a certain style, format, etc.

Fine-tuning is done by giving the Ai a prompt, an assumed user input, and a completion, of what the ai should output to complete the prompt. For example, if you wanted to fine-tune a model to make it answer all questions by saying "Hmmm, let me see: " at the beginning of every sentence, you would pass in the prompt, and for the completion you would right "Hmm let me see: " followed by the answer to the prompt. When a model is fine-tuned it is often trained on a vast amount of text data. Ai Dungeon was trained on about 40GB of text data. [Harris 2020]. The model for this project was trained on about 3.27 MB of text data.

The result of Ai Dungeon's use of Ai is the ability to make virtually anything happen within the stories it generates. It will understand what the user inputs and make a story out of it, and it will include any details that the player inputs. Hua and Raley, in their work Playing With Unicorns: AI Dungeon and Citizen NLP, discuss the many contributions of the game, how it works, and how it affects the study of Ai. However, as previously discussed the definition of a game is that it must have rules, conflict, and a quantifiable outcome.

While Ai Dungeon does have rules, there is seemingly no conflict when the user holds the power to do virtually anything. The game could take place in a medieval setting, but if the player writes "I pull out my smartphone" the game has no limits against this. This also means if the player is confronted with an obstacle, say a locked door, all the player must do is deny that it is locked in order for them to be able to open it. There are rules in Ai Dungeon but because of the limitless freedom it gives there is no conflict or outcome.

A game with conflict and outcome would deny the player the ability to use a smartphone if they do not explicitly have one. Furthermore, the game would deny the player the ability to open the locked door unless they had a key or opened it with an ulterior method. That is where this project attempts to succeed, giving the player limitless freedom but containing set rules and a set outcome for every action.

Approach:

The objective of this project is to be able to process any input the player does and make it affect the game world in some way. To do this task, OpenAi and GPT are used. The Ai figures out the player's intentions and writes a description of the results through two methods, Action Detection, and Description Extension.

Action Detection is used to predict the player's intended action. For example, if the player inputs "hike south" the Ai should process this information and come to the conclusion that the player's intended action is to move south. Then, Action Detection informs the game engine of this information and the game engine changes in-game variables as needed. In the previous example of "hike south" the result would be that the player's current location is changed to the location south of where they currently are. Any action, as long as it's feasible, can be input. An action is feasible if the subject of their action detection is in the player's inventory or the environment. The player would not be able to do "use cellphone" if they do not have a "cellphone" to use.

To make a possible outcome for every possible verb, Action Detection filters the verb into one of eight main categories: "see", "smell", "speak", "taste", "touch", "use", "move", or "attack". If it is unknown which category the verb should be placed into, the closest possible match is made. For example, the verb "criticize" is closest to "speak" since they both involve speech. If the game still cannot decide what category it is, the game will put it in a ninth "other" category.

The verbs "other" and "attack" require the player to make another input. This is to get a more accurate representation of the player's intentions. For example, the player may write "attack goblin" but in order to figure out what the user wants to attack the goblin with, an additional prompt is given. The "other" verb is used for miscellaneous verbs such as "dance", "spin", "light", or other verbs. The player is prompted if they would like to do an item to do this verb, to better get an idea of what the player wants to do. If the player wants to light a fire, asking for what item they use can make it more or less effective, such as the player using a torch to light the fire making the task more detailed.

Description Extension is used to output to the player the results of their action. This can be small things, like if the player wants to feel the floor they should be informed how the ground feels and any relevant details, this can also be for larger, main game mechanics such as attacking enemies or opening doors with keys, telling the player how they performed those actions.

Description Extension also takes into account the background and descriptions of different characters, items, actions, etc. This is to get a better, more detailed, and more realistic response. For example, if the player asks a goblin about the current state of the economy, the goblin should not give an exactly accurate answer on that topic unless it is explicitly said in their description that they have that knowledge of the economy.

To better get an accurate depiction of the player's intentions, there are three kinds of descriptions: location, dialogue, and action. Location is the details of environmental aspects. This can be the look, smell, etc. of the current location or other aspects as well. This is mainly used for verbs like "see", "hear", "smell", "move", "use", and "taste". Dialogue involves speech between the player and any characters present, it mainly is used when the verb is "speak". Action is what directly happens from a player's action, it is mainly used for "attack" and "other"

A single model utilizes OpenAI and GPT to do Action Detection and Description Extension. Firstly, the player's actions are input into the game in the form of text as (Verb Subject), or in quotation marks if the player is speaking. If the input is not in either of these forms, the input is invalid and the player is prompted to do another input. Action detection is used to figure out the intention of the user.

If needed, the player will be prompted to more about what they want to do, such as if they want to use a specific item. The item must be in their inventory or the environment. After this, the game changes any relevant variables if needed. Once the player's intended action is detected, the game should take into account any items, characters, locations, and backgrounds and then send that data to the Ai model for its other task, description extension. Finally, the game outputs the results. The end goal is for the player to do any feasible action and for that action to have a result. The whole process looks like this:

Input → Action Detection → Additional Query (If Needed) → Change In-Game Variables based on Action Detection results (If Needed) → Description Extension → Output Description Extension to player. A simplified version of the process may look like this:

Input → Action Detection → Query Additional Info → Description Extension

Finally, two main data structures are used for the game engine portion of the game. First is one with static variables, gameLocations. It holds the data for all locations. Each location holds items, characters, which locations the location is connected to and where, as well as all their descriptions. The data is held via dictionaries and lists, so that it is convenient to search for the various data each location holds.

The othe data structure is the gameState data structure, which ho,ds the current circumstances of the game. It monitors the player's curren location, inventory, and other variables of the player.

Pseudo Code:

//game_locations = data structure that holds static data for all locations, items, characters, and their descriptions. It has a get_description(location) function for returning the description of the location, get_characters(location) function for getting characters in the location and their descriptions, and get_items(location) for getting items and their descriptions for a certain area. EX: If there is a goblin and a rock in the forest then doing get_characters("forest") returns goblin and its description and get_items("forest") returns rock and its description.

//game_state = data structure that holds the player's current location, inventory items, and other required information.

//get_response(prompt) function for getting a response from the Ai.

```
action_detection(input):
    verb,subject=get_response(input)
```

```
		If verb == null or subject!=null
			print(Invalid)
		return verb, subject
	description_extension(verb, subject, using_item):
		characters=game_locations.get_characters(game_state.current_location)
		items=game_locations.get_items(game_state.current_location)
		If using_item==false:
	output=get_response(“You are at ‘{game_state.current_location}’
{game_locations.get_description(game_state.current_location)}’ You ‘{verb}’ ‘{subject}’ Characters:
‘{characters}’ -> Items: ‘{Items}’ ->)
	Else:
			output=get_response(“You are at ‘{game_state.current_location}’
{game_locations.get_description(game_state.current_location)}’ You ‘{verb}’ ‘{subject}’ Using:
‘{using_item}’ -> Characters: ‘{characters}’ ->)
		print(output)
	main:
	While game_is_running:
		player_input=input()
		verb,subject=action_detection(player_input)
		If verb == verb_that_uses_item:
			using_item=input()
			If an item in game_state.inventory():
				description_extension(verb, subject, item)
			Else:
				print(Invalid)
		Else:
```

description_extension(verb, subject)

Methodology

The model used for this project was GPT-3 curie. It was fine-tuned using about 3.2 MB of text data. The data was prepared by creating a fileWriter python script. The fileWriter script has 8 lists of verbs coinciding with the previously discussed 8 verbs. Within these lists are samples of similar vers. For example, the seeVerbs list contains verbs such as “look”, “examine”, “investigate”, etc. These 8 lists are combined into one larger list of possible verbs. The subject then is taken from a large text file of possible names and nouns.

The fileWriter script then mixes and matches these verbs and subjects and writes them as a prompt. Then, the completion of the prompt is based on which of the 8 ver lists the verb being used resides in. So if the prompt was “investigate apple” the completion would be “see apple” The model was fine-tuned on over 36,000 prompts. An example is shown below.


```
{"prompt":"run ava ->","completion": "move ava"}
{"prompt":"fly jovana ->","completion": "move jovana"}
{"prompt":"tackle marnie ->","completion": "attack marnie"}
{"prompt":"investigate chandria ->","completion": "see chandria"}
{"prompt":"rend atlee ->","completion": "attack atlee"}
{"prompt":"snuffle interpretation ->","completion": "smell interpretation"}
{"prompt":"skewer meena ->","completion": "attack meena"}
{"prompt":"steal felcia ->","completion": "touch felcia"}
{"prompt":"pull demetrio ->","completion": "use demetrio"}
{"prompt":"distinguish ramina ->","completion": "see ramina"}
{"prompt":"pull aprile ->","completion": "use aprile"}
{"prompt":"slash mckenzie ->","completion": "attack mckenzie"}
{"prompt":"pierce jayla ->","completion": "attack jayla"}
{"prompt":"axe-kick faron ->","completion": "attack faron"}
{"prompt":"kiss aly ->","completion": "taste aly"}
{"prompt":"snuffle kenosha ->","completion": "smell kenosha"}
{"prompt":"kick josephina ->","completion": "attack josephina"}
{"prompt":"ambush kanisha ->","completion": "attack kanisha"}
{"prompt":"beat-up utensil ->","completion": "attack utensil"}
{"prompt":"fly domique ->","completion": "move domique"}
{"prompt":"easedrop clip ->","completion": "hear clip"}
{"prompt":"covet briar ->","completion": "touch briar"}
{"prompt":"lunge elias ->","completion": "attack elias"}
{"prompt":"fire toby ->","completion": "attack toby"}
{"prompt":"hold aylin ->","completion": "touch aylin"}
{"prompt":"hit sharis ->","completion": "attack sharis"}
{"prompt":"snuffle tiera ->","completion": "smell tiera"}
{"prompt":"depart gossip ->","completion": "move gossip"}
{"prompt":"feel ichael ->","completion": "touch ichael"}
{"prompt":"axe-kick limestone ->","completion": "attack limestone"}
{"prompt":"lunge indiana ->","completion": "attack indiana"}
{"prompt":"clutch marquella ->","completion": "touch marquella"}
{"prompt":"overhear jamarrio ->","completion": "hear jamarrio"}
```

When fine-tuning, the model receives prompts and completions from a preparation set. The ai also runs a validation set that has its own prompts and completions. To test accuracy, the ai will predict what the completion for a validation set prompt is. If the ai successfully predicts the text of the validation set compeltion then it is a success. The ai is continuously tested with the validation set as it trains, with the hope being that when its done training it will receive a high success rate with the validation set.

The example is a couple of many lines of text used to fine-tune. The goal is to teach the model how to complete the text when given input in the form verb, subject. In the prompt text, the character "->" is used to denote the end of the prompt. This is the inform the model to stop searching for prompt text and begin completing text. Later, the "->" character was changed to the "@" character and used for both the prompt and description to better inform the ai when a completion is finished.

Below is an example of the result of Action Detection. The verb is converted to the closest possible verb and the subject is kept the same.

```
Prompt: bite rock @
Completion: taste rock @

Prompt: cartwheel south @
Completion: move south @

Prompt: push button @
Completion: use button @

Prompt: roundhouse-kick statue @
Completion: attack statue @
```

Description Extension, however, uses a different approach. To teach the model to better distinguish when it should perform its two tasks, prompts that should use Description Extension are surrounded by quotation marks. An example is shown below:

```
Prompt: "You are in a green forest." @
Completion: "You are in a green forest. Birds loudly chirp and the shrubs rustle in the wind. An assortment of mushrooms sprout
from a log, each bulbous head about to burst spores. A nearby river gently trickles in the distance and the trees
dance in the wind." @
```

In order to include characters and items, they are denoted with their respective title, followed by a list of characters and items to include. The "->" character is used to denote the end of the list of characters or items.

```
Prompt: "You are at the beach. Characters: John, Albert -> Items: seaweed, shell, bucket ->"
Completion: "You are at the beach. John, Albert, and you are baking in the sun. The smell of salt hits straight through you.
Albert uses his bucket as a helmet, making himself look like a british soldier. Seaweed clings to the rocks,
refusing to let go even as the waves crash onto them. A seashell, glimmering like glass, is wedged into a sand castle." @
```

To denote action a "*" character is used. Action usually requires a character using something, so to pass to the model the items being used in the action they are listed with "Using:" which is also ended with the "->" character. An example cna be seen below:

```
Prompt:"* You attack the golem. Using: hammer -> Characters: Golem $ A large golem made of stone." @
Completion: "You swing at the golem with your hammer. Its leg shatters off of its body. The golem crashes into the ground,
 creating a crater from the impact." @
```

Finally, to inform the model to do dialogue, a "%" character is used at the beginning. Additionally, to provide description about a character or item it is listed after the character or time with the "$" character as seen below:

```
Prompt: "%  Characters: Johnny $ Your friend. , Eric $ Your enemy. -> Hey what are we doing for lunch?" @
Completion: "'I was thinking of hitting that new pizza place,' said John. He and you used to get pizza when you were kids.
'I hate pizza,' said Eric. When you and John got pizza he would dunk your face into it and laugh." @
```

Results:

Below are some sample results of input and output to the ai. The model understands the users intentions and outputs the result to them:

```
(tag) C:\Users\Brent\Documents\Ai\final>python tag.py
Enter Your Name > Brent
Welcome to the game Brent.
> use stick
 Using the stick you prod and poke at the ground feeling around for anything that might be hidden
```

Here, the crawl verb becomes the move action, as a result, the player's location changes to the location south of their current location, the cave, and the output shows the outcome of this:

```
> Crawl South
 "You leave the cave and crawl to the forest. You are in a dense forest, the trees towering above you. The air is
damp and the smell of moss hangs in the air. A thick fog blankets the ground, making it difficult to see more than a few feet
in front of you. The birds chirp in the distance and the sound of running water can be heard."
```

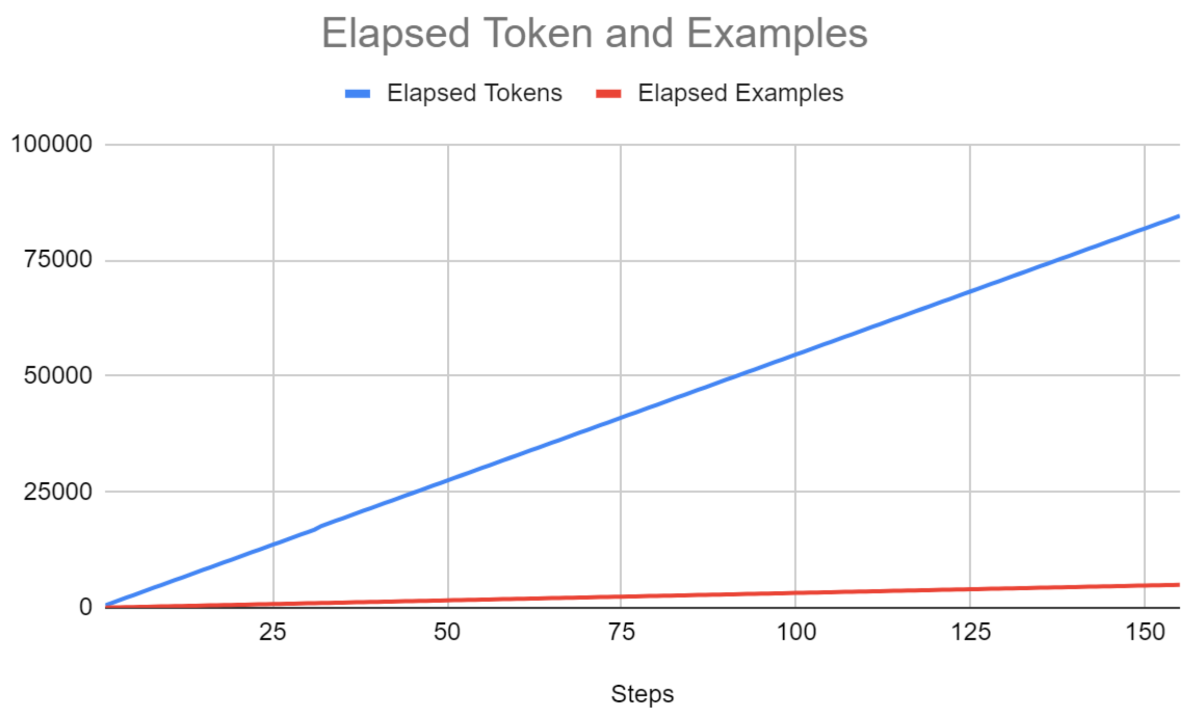
Dialogue works as well, with the two present characters, Bobokin and Bokoblin, interacting with the player.

```
> say im hungry
 'I'm hungry too' said Bobokin, rubbing his stomach. 'Let's find something to eat around here.' The Bokoblin nodded in agreement.
```

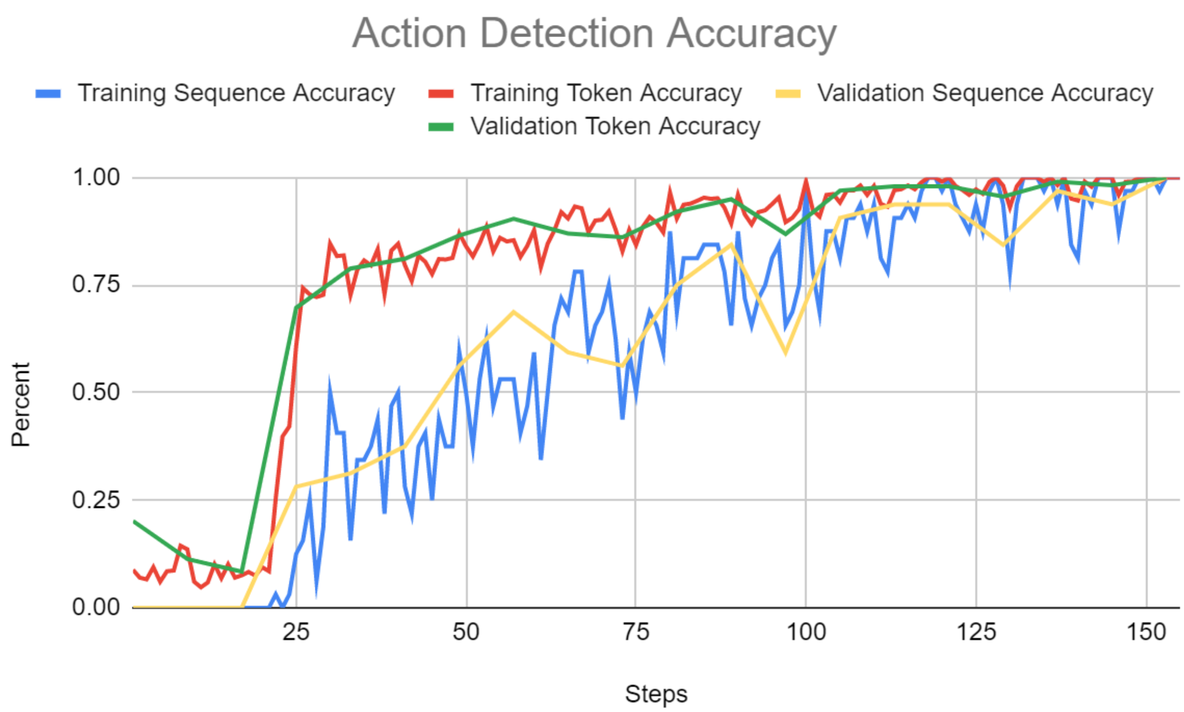
Finally, a realistic depiction of combat is done when the player decides to attack Bokoblin wiht a rock. The attack verb is used here.

```
> Crush Bokoblin
Use what to 'crush'?
> Rock
 You lift the rock and bring it down on the bokoblin It crumples under the weight of the rock and its lifeless body lies on the ground
```

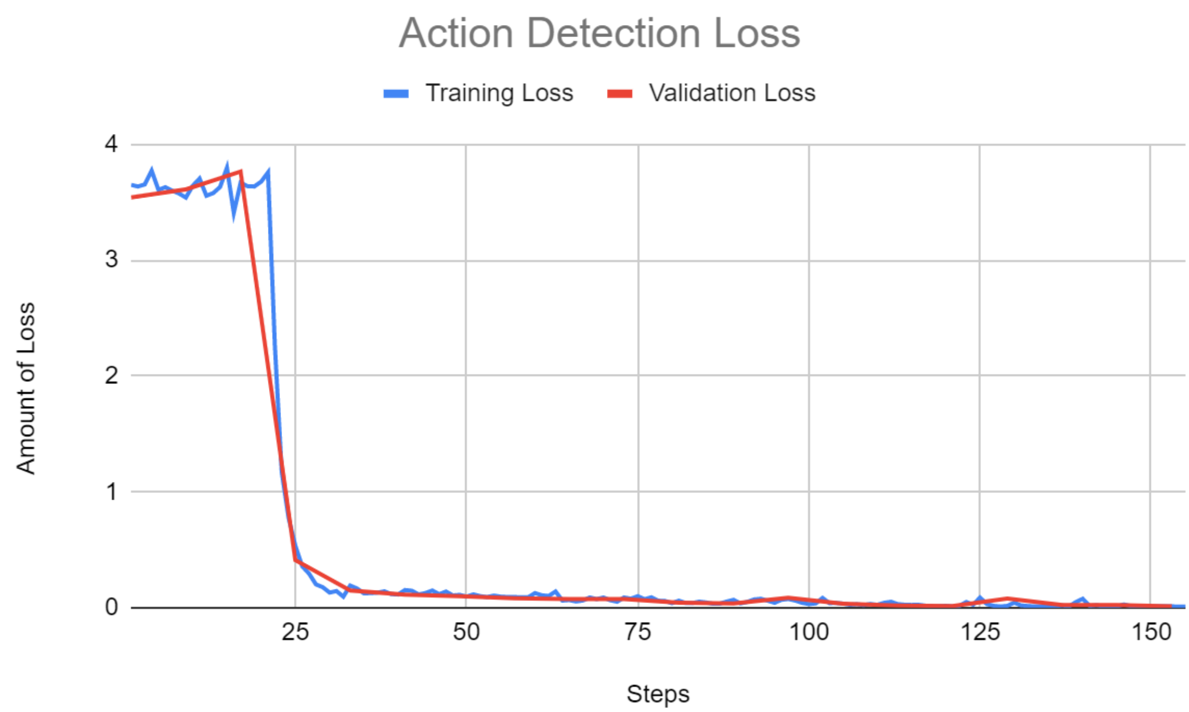
Below are the results of the fine-tuning done to the Ai model. This first graph is the elapsed tokens and elapsed examples. It shows how as the steps increase, how the amount of observed examples and observed tokens increase. Tokens are sub-words that the model breaks down words into to better get an understanding of intention.The takeaway from this graph is tat the model successfully went through many examples



The next chart shows the various accuracies of the fine-tuning. Training sequence accuracy is the ai's ability to predict all the words in the completion sequence for the training set. The same goes for the validation sequence accuracy, but for the validation set. The training token accuracy is the ai's ability to predict the next token in the completion sequence. The same applies to the validation set as well with validation token accuracy. What the results below tell us is that, at first, the ai could not consistently predict the next token or word in the sequence. However, as it continued to train it eventually was able to predict the next token and entire completion sequence for both the training and validation set.



The graph below is the loss for the training and validation sets. Machine learning algorithms have a loss function which measures errors between the predicted an actual result of a completion. As the model fine-tunes its parameters are changed to make it better at predicting the next word in the sequence. A model that is too complex, however, can end up following training too well, making it worse at learning new data and making it repeat mistakes it may have made in the past, such as not properly filtering out nonsensical characters. Monitoring loss can help prevent those erros from occurring, and as the graph shows loss decreased as steps increased.



Discussion and Future

As can be seen from the graphs and their explanations in the results section, fine-tuning the model was a success. The model was able to accurately predict the completion of the prompts given to it, getting nearly 100% near the end of fine-tuning. However, there was not enough time in order to complete the main goal of the project, that being integrating the model with the game and making a game with set rules, conflicts, and quantifiable actions, and the player can perform any feasible action they wish. There is a game, but in its current state it has no end state or required tasks. Visiting this project in the future could mean adding more story content to the game and expanding the game world. Additionally, having more computing power as well as time it may be possible to train the model to better recgonize more words. Occasionally, there are some words which could feasibly be put in one of the eight verb categories but is put into the “other” category. In order to have a viable project to present, text-davinci-003 was implemented into the game. While training results came back successful, the model was unable to be properly ported into the game in time for turning the project in.

I believe that with text-adventure games and ai, the possibility of a game with set rules and quantifiable outcomes but actions only limited to the game world and imagination could be a reality. Most likely, to fully optimize a game like this not only would the model have to interpret the text but it would have to change variables and make new pens as needed. For example, with my current model the player can light a torch, but the game would not recgonize a dark area as now being lit unless that was functionality explicitly coded in. A truly reality-like game may create a variable for the torch of lit=true. Then it may make a rule that all wood objects can have a lit variable as well, making rules consistent throughout the world. Then if anything is dark, dark is now equal to false.This functionality, however, most likely is not currently possible. Essentially, you would need code that can be interpreted and changed by the ai, as well as text input. As advancements in technology increase, this may become feasible but it is currently unknown.

Work Cited:

Salen, K., & Zimmerman, E. (2010). Rules of play: Game design fundamentals. The MIT Press.

Hua, M., & Raley, R. (2020). Playing With Unicorns: AI Dungeon and Citizen NLP. National Endowment for the Humanities, 14(4), 1–33. Retrieved April 24, 2023, from http://www.digitalhumanities.org/dhq/vol/14/4/000533/000533.html.

R. Fan and Y. Liu, "Research on Augmented Reality Interactive Games," 2011 Third Pacific-Asia Conference on Circuits, Communications and System (PACCS), Wuhan, China, 2011, pp. 1-3, doi: 10.1109/PACCS.2011.5990152.

Harris, J. (2020, January 9). Creating the ever-improvising text adventures of ai dungeon 2. Game Developer. Retrieved April 25, 2023, from https://www.gamedeveloper.com/design/creating-the-ever-improvising-text-adventures-of-i-ai-dungeon-2-i-#close-modal

What is natural language processing? IBM. (n.d.). Retrieved April 25, 2023, from https://www.ibm.com/topics/natural-language-processing