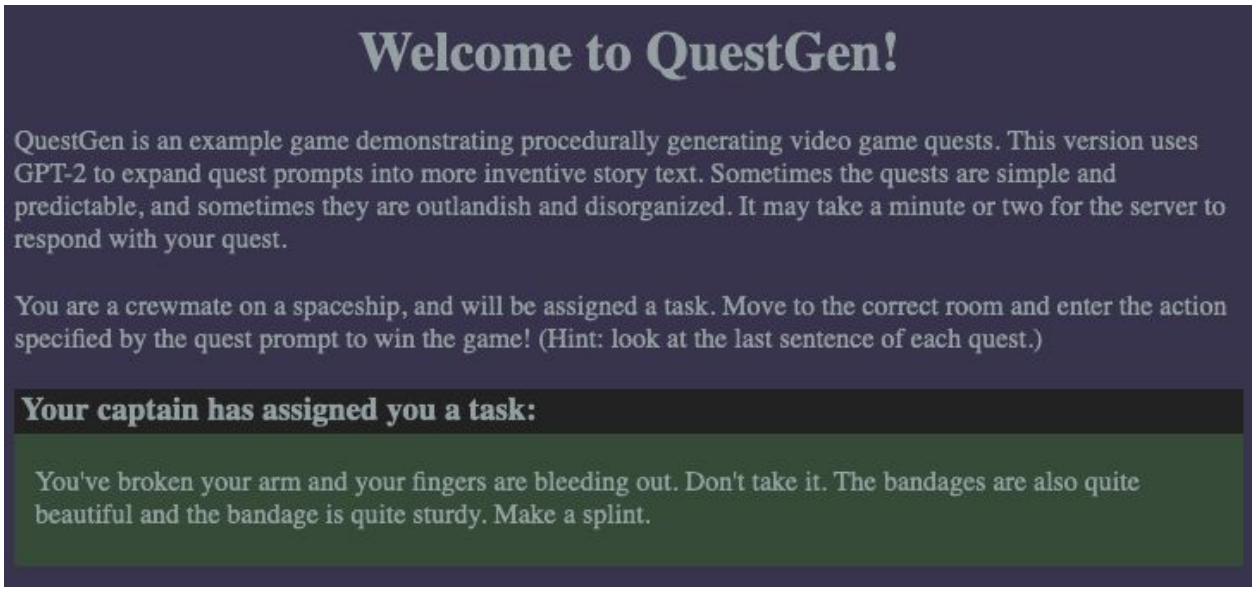
Austin Shafer
Suyash Jain

QuestGen: procedural generation of game quests

https://github.com/amshafer/QuestGen

QuestGen is our attempt at generating creative and expressive video game quests using simple grammar techniques combined with advanced GPT-2 generational models. It was intended to be a toolkit that could be paired easily with a variety of game engines, but the final version we settled on is an enhanced version of techniques used in existing games. We created a simple game to demonstrate our findings, in which the player is assigned a generated "task" (a quest) and must complete said task to win the game.



Many video games have simple procedural quest generation systems. These create small missions for the player, and usually involve simply sending them to a location to defeat an enemy. Our expectation was that we could take a simple system such as this, and extend it using modern machine learning to create more dynamic and less predictable quests. We wanted to wrap this in a toolkit so that others could use it without having to relearn everything we did.

In reality we found that creating such a toolkit is extremely difficult, and although we did successfully extend simple generation with GPT-2, using machine learning in most games is too computationally expensive. A procedural generation system is going to be very closely tied to the game and the game engine. Tracery is a good example of how this can be done well, because of how lightweight Tracery is. Once machine learning is introduced, it becomes near impossible to run everything on the average computer.

Another problem that we had to deal with was using GPT-2 instead of GPT-3. GPT-2 is very powerful, but requires fine-tuning in order to work exceptionally well. Unfortunately, there is no corpus of video game quest text, and making one is not easily feasible. This meant that we had to use the raw GPT-2 model from openai, ruling out any possibility of using solely GPT-2 without any sort of structure. GPT-3 by comparison is designed for "few shot" generation. i.e. you give it one or two examples and it can generate stories of that format. This is much more ideal to our needs, but unfortunately GPT-3 is not publicly accessible yet.

What unsuccessful designs did you explore? What made you decide not to pursue them?

Our initial research was into teaching GPT-2 to generate quests on its own. This quickly was ruled out due to a lack of fine tuning data. Our next (and opposite) approach was using only tracery as we have before, but this yields a much smaller set of possible output texts. We wanted to stick with GPT-2 for the randomness it would provide our passages.

A technical design issue we experienced was deciding where the GPT-2 model should be run. Ideally it would be run on whatever computer is running the game, but this cannot be guaranteed to be the case. GPT-2 is extremely memory hungry and most devices would not be able to run it. For this reason we realized the machine learning portions of our prototype would have to be run on a server, and be queried from the game. This allowed us to choose a lightweight web client for the game, and shaped the way we thought about expanding passages. It also limited the size of the passages we could produce, as waiting for the server took an unacceptable amount of time with longer passages.

Therefore we settled on using Tracery to generate a small quest, and using GPT-2 to expand it into a larger text. This was done by having Tracery make a 4 sentence passage, and using each sentence as a prompt that GPT-2 would use to generate a sentence that followed it. One huge benefit of this approach was that we could guarantee the quest would have coherent structure generated by Tracery. This is usually a problem when using ML generational techniques. It also made it easy to incorporate into existing games: the game selected a win condition, generated a simple structure with a grammar, and used ML to add color and expand it. This actually worked pretty well, but the sentences GPT-2 emitted were somewhat unrelated and really threw off the readability of the text.

Our final design is based on a refinement that leans into GPT-2's strengths: using GPT-2 to complete sentence prompts instead of creating new sentences. GPT-2 is most useful when trained on a dataset or used for autocompletion given a word or sentence. Given that we didn't have a dataset for fine tuning, we would most likely have success using it as an autocompleter. We changed our program to create a set of "prompts": half sentences that are somewhat structured but still vague. We gave these prompts to GPT-2 to expand into full sentences, and combined them to make the final passage. This keeps the benefits of coherent plot that generating with a grammar provides, but prevents GPT-2 from getting sidetracked and creating sentences that sound "too random".

What did you learn from working with playtesters?

When the playtesters started playing the game we received a lot of feedback from them regarding the gameplay and the way they were able to go through the story. One of the first suggestions that we received was that there were substantial issues when a certain action was taken and the output didn't reflect anything giving the suspicion to the user that the game has frozen or not responding. We realized that the user needs to see that the action wasn't useful so we applied that to our model.

Another major factor that the playtesters caught was that even after performing an action the user was able to see that action again but it was a part of the gameplay as a user can perform as many actions as they want when they play inside a room.The ways in which the stories progressed and how the users reacted also gave us the idea to improve the storyline.

Another main feedback that we received was that the UI was a bit blend we worked through on it and were able to make improvements.The playtesters played the game thoroughly and used each action that they can perform to test the output from the quest.

What technical challenges did you face, and how did you overcome them?

The largest technical challenge by far was getting GPT-2 to run. My laptop was not powerful enough to run it, so I needed to use a server. Additionally, we had to worry about the players running into the same problem. The odds are they do not have all the GPT-2 dependencies installed, and they also might not have a powerful enough machine. The only server powerful enough that I had access to does not have a GPU, and as a result our GPT-2 generation runs extremely slowly. Because of this, we decided to make the GPT-2 generation part a web server that could be queried from anywhere. The playable game is a simple html file that makes HTTP GET requests to an instance of our code that I am self-hosting publicly. The game waits for a minute or two for the generated quest to be constructed, and then proceeds to display the game. Luckily simply running our existing code on a larger server with a powerful GPU would increase the performance to more acceptable levels, such as using an AWS node.

Additionally, due to the computational expensiveness of GPT-2 we limited ourselves to only using the smallest (124M) model available in order to prevent excessively high wait times for quest generation. This also has the downside of producing lower quality text. Using our unchanged code with one of the larger models on a powerful GPU would provide significantly faster and higher quality responses.

A minor technical challenge was that the tensorflow package was broken on the operating system of the server I was using. I don't own this server and could not change OSes, and running a virtual machine made tensorflow (which GPT-2 uses) even slower. I had to fix tensorflow (and submit my patches to the distro) to build a compatible version for GPT-2 to work.

How does your final prototype differ from your initial concept?

Our original goal of creating a perfectly reusable toolkit fell through, and we ended up producing a more traditional game. Although we did not produce a widely reusable code library, we think that our approach in using GPT-2 for story grammar sentence expansion has the possibility of being used elsewhere. Especially if used with a capable cloud server running a few-shot generational model such as GPT-3, we think this design could be useful in real world games. Our final prototype is really a proof of concept of how this approach could be implemented.

What grade do you feel the execution of your project deserves based on the rubric you submitted in your concept document?

We feel that we accomplished most of the goals we set out in our proposal, and would argue that we deserve full marks. We did produce an example of optimally using generational models to generate quests, but we did not create the reusable toolkit we intended to. For not creating a library that others could use, we could understand losing a couple points for not achieving this goal. We argue that we deserve full marks because we did set out to create such a toolkit, but found that there was no clean way of doing so that would be widely usable in other games. Despite realizing that our goal was flawed, we demonstrated different techniques for extending the simple approaches used in many games, and documented the tradeoffs that developers can expect if they go down the same road we did.

Play testing

1 . You are a crewmate on a spaceship, and will be assigned a task. Move to the correct room and enter the action specified by the quest prompt to win the game!

The first clue that the captain stated was “The emergency lights are flashing, and the car is being towed. "I'm just really pissed off," said the driver, who asked not to be identified. The room is lit up red , and a man in a white suit is sitting on a stool. He is wearing a white shirt, a black tie, and a black tie. You need to plug in the wires to turn the alarms off.” The user chose “move-to electrical” as the action they would like to take. This moved them to electrical and they had another set of possible actions from which they chose to “Plug in the wires” . The reason they gave for this decision was that since the hint states that the last sentences of each quest helps in the completion of the quest. This action stated that they won. That was the first quest and that was won by the user.

The next clue was “The spaceship needs to be able to make a landing on the surface of the planet. The ship will be able to make the landing on the surface of the planet. You replaced the broken component and need to pay for the new one. Swipe the credit card.” Since this required a credit card the user first chose to “clean the counters” Which didn’t result in another sub action which gave the user the impression that the action that they chose was not the one required to fulfil the task. The user then went to choose “rebuild the chair” which again resulted in no further

actions. The next choice for the user was “move-to cafeteria” Which showed them the same actions as they had when they started the quest. Next Up the user chooses “ move-to administration” which moves the user to administration. Now the actions changed a lot and the user selected “swipe the credit card” Which completed their quest and the user won.

Feedback from User 1 - "The options need to have a say when there is a selection of wrong action or atleast if you select that action and that doesn't work that action should be removed from further actions . Also when you go to a particular place and you have moved to that place that should no longer be in actions . "

The new quest was “The main computer is under the control of a person who has been given the opportunity to use the computer for a period of time. (2) The person who has been given the opportunity to use the computer for a period of time is entitled to a copy of the computer and the person who has been given the opportunity to use the The data needs to be saved and made available to the public. Backup the sensitive emails before something happens to them” . The user selected “move-to cafeteria” . This moved the user to Cafetaria. After this action the user went for ”clean the counters” as his next action resulting in no change of actions . The user then selected “move-to administration” which moved the user to administration . The next action was” backup the emails” . Which completed the required task for the quest and the user was shown the win message.

Feedback : When there is an action that results in no action we should remove that action and only give remaining actions.

The next quest was “The ship has lost power and is in a state of emergency. "We've had to evacuate the vessel and we're going to be in a state of emergency for a very long time," said a senior official from the Navy. Don't take too long, too much time without power . The power of the sun is like a light that shines on the world. The oxygen won't last forever . The oxygen won't last forever. Restart the generator” . The user selected “wash the dishes” as his actions resulting in no change of subactions . The user was just going through every action available before changing the room. The user next selected “clean the counters” as his next action which didn’t change the outcome. Next “rebuild the chair” was selected as the action which didn’t show any promise as well for the user.”fix the floorboards” was next selected which didn’t result in any specific subaction . The user again selected “fix the floorboards” as next action which didn’t receive any reaction as well.The user then selected “move-to electrical” as his next action giving him new subsets of actions . The user selected ”Plug in the wires” as their next action resulting in no outcome. The next action was ”Flip the switches on” which again gave no change in the outcome. The next action was “Restart the generator” which resulted in the user winning the quest.

Feedback : It takes more time to have a response from the server for a new quest. There has to be an indicator for wrong selections and once an action is taken it should not be available for selection again.

Deliverables

Analysis:

1. Play “FATE: undiscovered realms” and write a 500-1000 word analysis of the experience playing quests

FATE: undiscovered realms is a small game that is very similar to Diablo, and many other dungeon crawlers. You stock up on gear in the local town, travel deep into a nearby infinite dungeon, and fight monsters as part of questing or exploration. Most of the content in FATE is procedurally generated, which leads to some very high replayability. There are hundreds of weapon and monster types, with thousands of possible enchantments and bonuses. The dungeon scales infinitely, meaning that the stronger a player gets the deeper they can go to still find challenging content. The quest generation system is of particular to us since we are making our own generation system.

There are a number of quest giving npc's in the "home" town that you start in. If you go to one of these npcs they will print out a block of text outlining the task they want completed. Usually this involves finding a monster at a certain level of the dungeon, or retrieving a long lost heirloom. The player is rewarded with experience and sometimes an item for successfully completing the quest. Although it is relatively easy to tell the general pattern of some of the quests, there are so many quest formats, items, monsters, and rewards that it's hard to get bored of the options you are given as a player.

FATE's quests are clearly generated using a simple grammar. There is usually a high level "plot" that is chosen, and the details are filled in randomly. FATE has a very large space of possible monster and item names, and even more types of monster/item classes. The quests feel like simple Dungeons and Dragons quests, where the actual quest isn't meant to be that interesting but is instead meant to give you a reason to explore the world and find some treasure. Loot is one of the most important game mechanics of FATE, so it makes sense that the developers would want to provide a consistent method of rewarding the player.

One of the main advantages of FATE's system is that it is very lightweight and easy to incorporate into any engine. The engine needs to randomly select a quest type, generate a few items or monsters, and record them as the win conditions for the player. Generating the quest text after this is very easy, and text can be displayed in almost any form in any type of game. This seems to be the approach chosen by a lot of games since it is relatively easy to make, is not computationally expensive, and generates coherent and predictable passages.

Overall FATE's system is very fun to play with, and it has admirable tradeoffs from a developer's point of view. I used to play this game a lot when I was little, and it was fun to return to it and see how it held up. It is interesting that the quality of random quests in most mainstream games has not changed for the most part, and modern generated quests are no more complex than FATE's

2. Take notes on the format of quests. List some common quest patterns.
The format is a passage containing multiple non-terminals which are expanded to produce the full text. It is similar to a very simple 1-level deep tracery grammar.

There are some high level "classes" of quest, each of which has multiple possible passage formats. i.e. kill a monster, retrieve an item, track down a named enemy, etc.

Some common patterns include:
"Help a monster has <been terrorizing us|stolen my item|...> and has fled to level <num> of the dungeon. Can you go kill him for me? I'll give you <item>"

"I lost my <item> in the <num> level of the <dungeon>. Can you go find it? I'll give you <num> gold pieces"

"I have a herd of <monster> that I need transported to <level>. Can you take them there?"

(These have been paraphrased and/or simplified)

3. Play radio missions in “City of heroes” and compare them to the quests in FATE in 500-1000. How are they different? Can you tell which “City of heroes” quests are randomly generated or hand written?

City of Heroes is a mmorpg that lets the player create a superhero with a custom set of powers, and pursue missions with other players online. They can explore multiple cities, save citizens, fight raid bosses, and even fight other players. Enemies are randomly spawned, or are placed in predefined locations as part of hand-crafted missions. There are many large storylines that have been designed by humans that can be considered the "main plot" of the game. Like many rpg's, City of Heroes has a sizable amount of randomly generated content. Surprisingly, the amount of randomly generated content is smaller than much smaller games such as FATE.

The random missions in City of Heroes are limited to the "police scanner" missions, also known as "radio" missions, along with a small number of other random encounters. The player can check a police scanner for nearby criminal activiy, and get told of a building that is under attack. When the player enters that building, they are effectively put in a small dungeon and can fight through all of the enemies to win extra experience. The main purpose of these missions is to make grinding a character up to a certain level easier by supplementing the experience recieved from enemies with quest experience. Other random encounters include bank robberies and burning buildings, where the format is similar but the player may have to save some civilian npc's along the way. Most of the missions I played were almost identical, and I quickly felt like I had seen all possibilities.

One of the most striking differences between this and FATE is that FATE has a significantly larger space of possible quest actions. City of Heroes’ missions all follow the same pattern: go

to a building, enter it, and defeat the enemies inside. The reward is always experience, there is no expansive loot system like FATE has. This makes the procedural content become boring and stale rather quickly, and makes it feel more like a chore.

When we selected City of Heroes I thought it had a generation system on par with FATE, just in a different genre. I figured contrasting the two would be interesting since they are achieving similar goals but in two very different genres. City of Heroes has a large open world with many concurrent players and must manage keeping them separate. I think the difference actually comes down to the focus of the two games: City of Heroes had a larger development team focused on the hand-made quests and FATE was made by a small team that focused on procedurally generating everything. They are both fun games, but one used random content creation significantly more than the other.

What is interesting is that the procedural quests in FATE are just as fun, if not more, than many of the hand-written missions in City of Heroes. I think this is an interesting story about how focusing on narrative generation is good for business, not just for the fun of the players. Significant man hours in an mmorpg can be saved by content generation, and the developers of FATE understood this.

Research :

1.
Learning to Speak and Act in a Fantasy Text Adventure Game :

The main theme of the paper was that most of the text adventure games that we play focus more on the statistical regularities of the language. Ignoring most of the virtual world in which the game exists. The premise on which the author's game works is that with the use of all the details inside the world the language predictor can be more accurate in the text generation over standard statistical models.This created LIGHT which is a multiplayer text adventure world which studied dialogue between players, models and the world . The model was trained based on the large dataset which had character driven human centered dialogues , interactions, actions and emotions.

The model was checked for its reaction to a situation where it could speak and act according to the environment and dialogue from the players.The model was based on BERT which is a state of the Art language model.The standard language models take the last few utterances from the dialogue history and create the next steps based on that. Several approaches for this text generation from the model exist but similarity with human dialogues is far fetched. This gives us a sense that the learning is either from action or dialogues . The LIGHT uses both together.

The major advantage of the author's setup is the amount of data that they have to train their model . The data was crowdsourced to accurately relate to human-like actions and dialogue the method used for the collection of this data was very impressive as the contributors provided locations for the game from various categories and they were asked to describe the location

along with backstory and some connected location. The data was collected and randomized to generate the world. Objects, Characters, Actions were also crowdsourced.

The learning models were used in two ways ,one was ranking models that output the maximal scoring response from a set of potential candidate responses and generative models that decode word by word. Each dialogue was considered a candidate for the player's next response and the ranking was based on the current environment and how the users interacted with the environment.The author also compared the performance of the best models to human performance on each of the prediction tasks. The BERT pretrained language model was used for the tasks of dialogue and action prediction.

For the task of action generation, the author used the set of candidates for ranking models to rank the true action sequence against the constrained by the set of valid actions. The environment data provides the best performance for both retrieval and generative models.

2.

Toward Automated Quest Generation in Text-Adventure Games

"Automated Quest generation games using narrative in language to help us understand human perceptions of creativity and what it takes to replicate this through computational models is natural". The Author understands that automated text generation games using computational models is a very natural way of understanding the creative human mind. The text adventure games consists of two parts 1) Structure of the world including the layout of rooms, textual description of rooms, objects, and non-player characters; and (2) the quest, consisting of the partial ordering of activities that the player must engage in to make progress toward the end of the game. Generating a quest is not an easy task as there has to be a coherence between the actions and the task performed by the user to achieve an end goal.

In a text adventure game this is even harder because of the natural language that is used as an input. The Author outlines the importance of grammar and outlines that the potential output space is combinatorial in size because of natural language. The user is limited to text which could affect the perception of creativity. The author uses a textwork framework for this dilemma . The way in which the quest is generated is that the author's game has a big database for the tasks and the related tasks . The user is asked to navigate through the environment and take actions which lead to the quest completion. The model used for content generation is markov chain along with neural language based models.

Each action will affect the probability of future actions. This gives way to the Markov chain which utilizes this pattern perfectly.Neural language models, designed to predict an element of a sequence conditioned on a given number of prior elements. The weighted graph is used for quest generation from a large base of quests and this generates the probable action for the completion of the task.

The dataset is important in the markov chain generation as it forms the basis of the graph. Each action for a node and the weighted connections of nodes create a quest. The model generates edges that are not unrealistic based on the given task thus making a completely coherent quest. The instruction generation or action is based on subgraph mining and prior generative methods based on probabilistic graph walks. Combining the use of only conditional probability leads to select few actions related to the task and never reaches an incoherent end.

The first language model uses a simple 4-layer LSTM . The dataset is used for training this model and then the end of the task is generating a token which marks the end of the quest. This trained model generates quests and actions . The actual task is generated by GPT-2 model and fine tuning the pre trained parameters.

After this step the quest is generated in the game world using knowledge graphs which are the object and action graphs. The object graph is set up for the objects placement in the environment The action graph was designed to prevent the player from conducting illogical actions.