The King and the Crowd: Process Doc

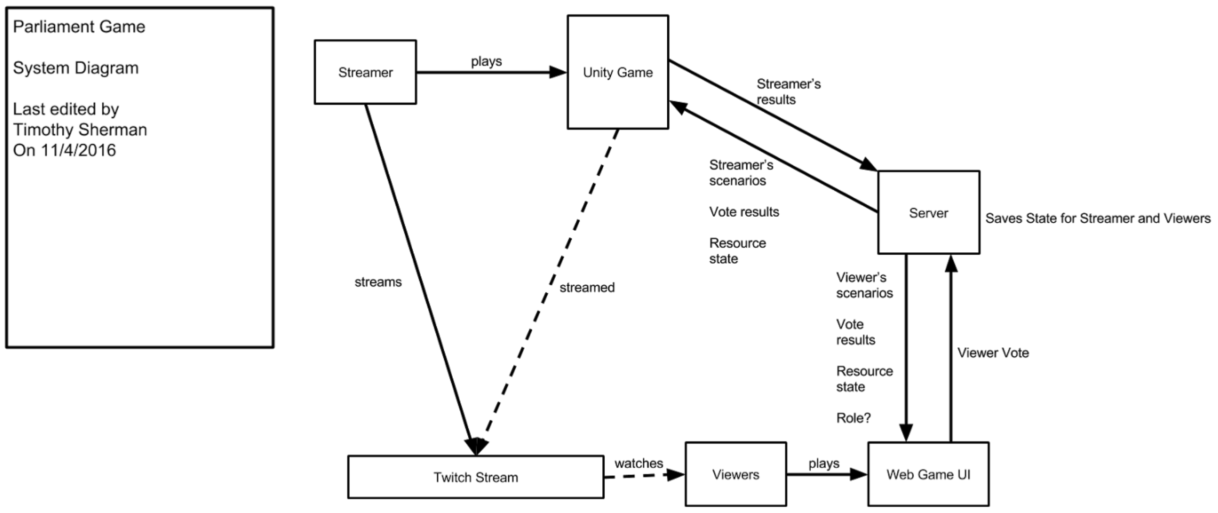
Team Parliament: Tim Sherman, Albert Yang, Ken Situ, Barun Kwak, Cole Gleason, Bobbie Soques

05-499/899C: Twitch Plays Game Design: Crowd and Cloud for Distributed Play | December 14, 2016

Beginning Technical Development

From the start, we decided to use Unity to develop the game, since it's the platform that our developers were most familiar with. As our game was going to be mostly text with some 2D art, we didn't need the 3D support that Lumberyard provides. We decided that the audience would need an interface separate from Twitch chat, to see their events and make their decisions, and to keep chat free for the streamer and audience to interact. We also that this interface would be a web client of some kind, so that the streamer could simply provide a link in the chat for the audience.

We created a system diagram which showed our initial design for the underlying technical structure. We knew we would need a streamer interface (Unity), an audience interface (web client), a way to communicate between the two, and somewhere to store not only the game stats, but also the events and the choices the streamer and audience had made.



Above: The original draft of the system diagram, showing each part of the game system and how they communicate.

One of our team members who had used it before suggested we use Firebase to save the game state, rather than building our own server using AWS. Firebase is "a realtime database, which provides an API that allows developers to store and sync data across multiple clients,"[1] and appeared to come already developed to work as a plugin to Unity. Unfortunately, it turned out that Firebase had been updated and the Unity plugins were all obsolete, and so much of the technical development then became focused on getting Unity, Firebase, and the web client to talk to each other.

[1] https://en.wikipedia.org/wiki/Firebase

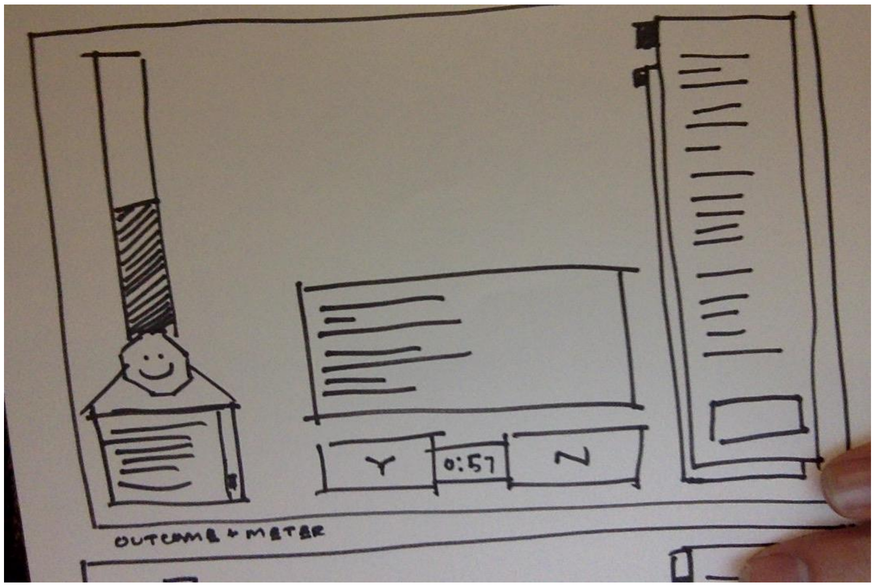
Beginning Game Design Development

While the technical parts of the game were being developed, we spent many meetings discussing the design of the game. We talked about how to make the audience feel involved and like they had a stake in the game's outcome, how to create tension, how choices should be presented to the streamer and audience, and how the stats should affect those choices. However, due to the difficulty developing the technical part of the project, we never were really able to test our designs, and so iterated in concept only.

We began by exploring our influences, and talking about how we might represent the state of the game world, how we could and/or should divide the audience into different groups, how a voting mechanism might work, the atmosphere of the game, and many more questions. Eventually we settled on the overall design we have now. The game contains three ruler stats (Military, Magic, and Diplomacy) which the streamer cares about, and three kingdom stats (Happiness, Wealth, and Power) which the respective audience classes care about. The streamer's events have multiple choices influenced by stats, but the audience's events have simple yes/no decisions that have to be made within a limited amount of time.

We wanted the game to be replayable, and feel different every time. As such, we focused on creating multiple "lose" states rather than a hard "win" state, and decided that a goal for each audience "class" could be to try to end the game only when their own kingdom stat was the highest. These design decisions, we hoped, would encourage streamer and audience to play as long as they like and continue to interact, rather than focus too hard on "winning."

At this point, we felt able to begin creating sketches and ideas for the streamer and audience interfaces. One of our team members came up with an assortment of designs for the audience interface, which needed to include four main elements: the event text, the yes/no options, the class resource, and the decision timer. We also thought it would be neat if the Twitch chat and stream was embedded into the web client so that the audience didn't need to toggle back and forth between the stream and the web client, and it would also be useful to have a chat specific to that audience's class.



Left: Sketch of the audience / web client interface. On the far left, the resource meter, with a record of past choices and outcomes below it. In the middle, the text display, with the Yes and No buttons beneath, and the decision timer between them. On the right, the Twitch chat and within-class chat, which can be toggled between. The empty space in the middle is where art or the Twitch stream would go.

Design Challenges and Interim Presentation

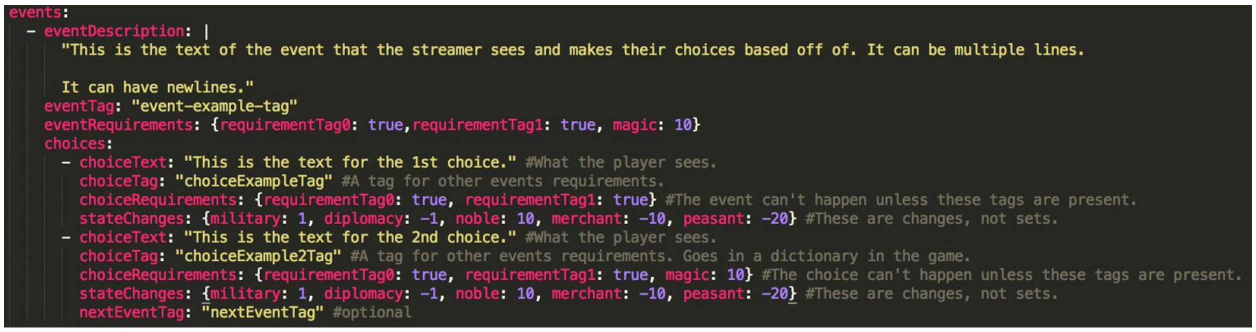
One problem we encountered was how to handle the decision cycle of the game. We knew we wanted different events for the streamer and for each audience class, and that some events might affect the others. For example, the king's decision to affect taxes could trigger a Tea Party-esque event for the peasants. Since we had decided on a procedurally generated story rather than a predetermined order of events, we had to decide how these events would be presented to the streamer and the audience. Should it be sequential, so that the streamer can talk to the audience without worrying about their own events, and vice versa? Or should the decision-making be simultaneous, which is more chaotic but also more engaging?

We presented our project so far to the class, as well as our issues with the decision cycle, and received valuable feedback. We added a history of past choices and outcomes to the audience interface, and the option to minimize the kingdom stats visible in the streamer interface to increase audience uncertainty. We also got new ideas on how to best develop the game in keeping with our idea of it being a relaxing "in between streams" game, by providing the streamer an option to "retire" once they'd passed a certain time, and increasing the likelihood of them dying once they passed that threshold, rather than having them die early on in the game. Unfortunately, these ideas didn't make it into this proof of concept, as it isn't designed for full play sessions.

We cemented our idea of having art to go with the text, to help provide building blocks for the streamer and audience to build a narrative. We also decided to go with simultaneous decision-making, which was the event-handling option the class supported almost unanimously. After the class's feedback, we settled on one of the web client designs that had been sketched earlier (pictured above).

Implementation

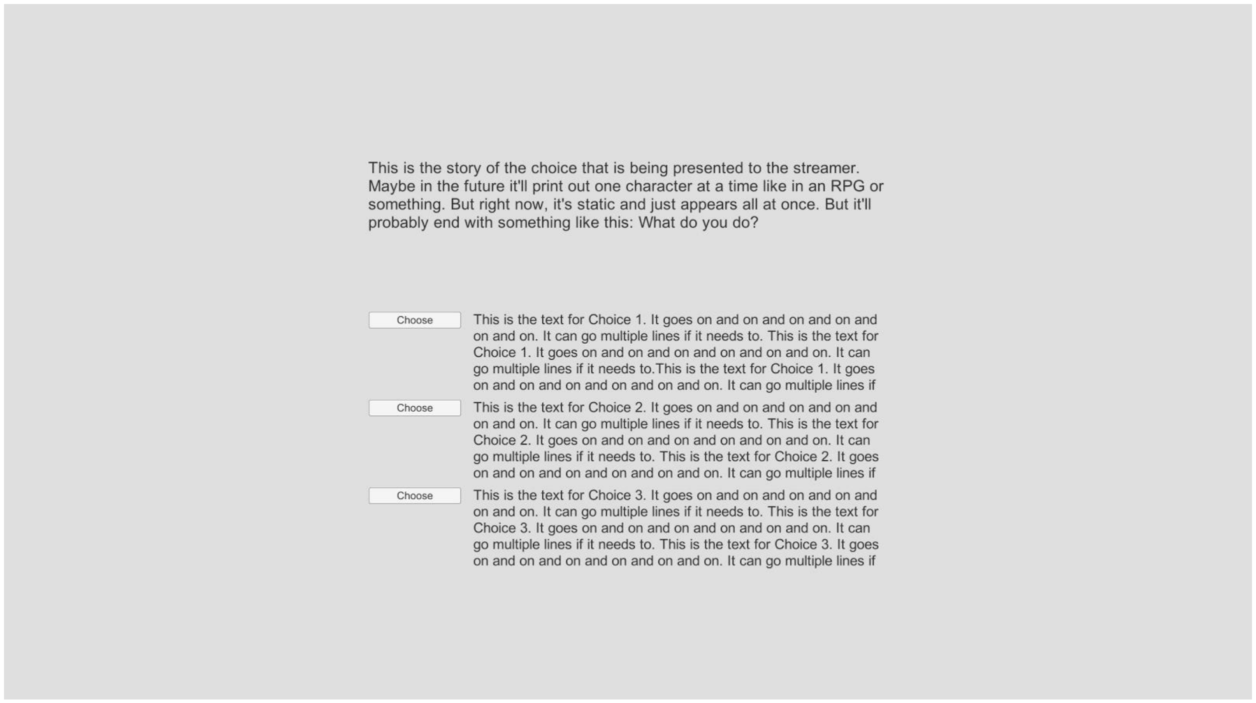
At around this time, we started working on taking the events we'd brainstormed and putting them into a format that could be loaded and displayed in the Unity and web clients. After some research and discussion, we decided on using YAML formatting for events, which unfortunately necessitated a bit of a learning curve on both the writing and loading ends. We also started developing art assets and trying to get them loaded into the Unity client, which presented our artist, who had never used Unity, a learning curve as well.



```yaml
events:
  - eventDescription: |
      "This is the text of the event that the streamer sees and makes their choices based off of. It can be multiple lines.

      It can have newlines."
    eventTag: "event-example-tag"
    eventRequirements: {requirementTag0: true,requirementTag1: true, magic: 10}
    choices:
      - choiceText: "This is the text for the 1st choice." #What the player sees.
        choiceTag: "choiceExampleTag" #A tag for other events requirements.
        choiceRequirements: {requirementTag0: true, requirementTag1: true} #The event can't happen unless these tags are present.
        stateChanges: {military: 1, diplomacy: -1, noble: 10, merchant: -10, peasant: -20} #These are changes, not sets.
      - choiceText: "This is the text for the 2nd choice." #What the player sees.
        choiceTag: "choiceExample2Tag" #A tag for other events requirements. Goes in a dictionary in the game.
        choiceRequirements: {requirementTag0: true, requirementTag1: true, magic: 10} #The choice can't happen unless these tags are present.
        stateChanges: {military: 1, diplomacy: -1, noble: 10, merchant: -10, peasant: -20} #These are changes, not sets.
        nextEventTag: "nextEventTag" #optional
```

Above: An example event in YAML formatting. We used this format for all our events, including audience events, which did not have choiceRequirements or more than two choices. Streamer events could and do have more than two choices.



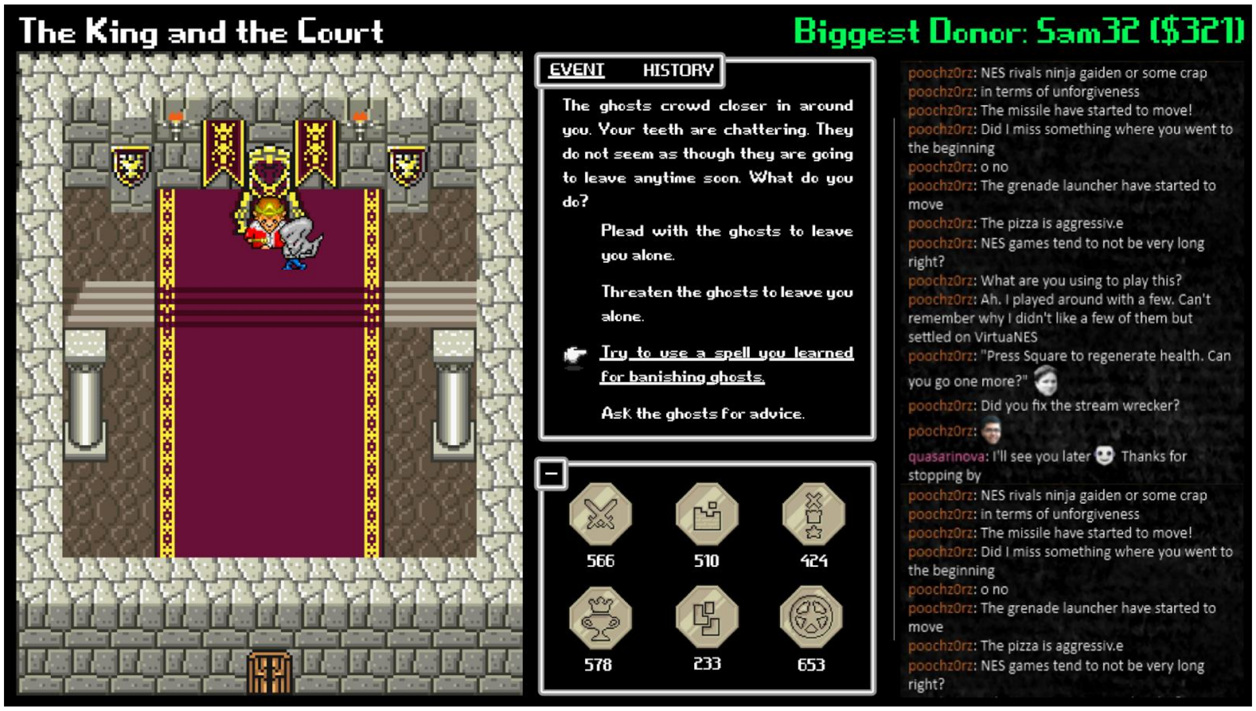
This is the story of the choice that is being presented to the streamer. Maybe in the future it'll print out one character at a time like in an RPG or something. But right now, it's static and just appears all at once. But it'll probably end with something like this: What do you do?

This is the text for Choice 1. It goes on and on and on and on and on and on. It can go multiple lines if it needs to. This is the text for Choice 1. It goes on and on and on and on and on and on. It can go multiple lines if it needs to.This is the text for Choice 1. It goes on and on and on and on and on and on. It can go multiple lines if

This is the text for Choice 2. It goes on and on and on and on and on and on. It can go multiple lines if it needs to. This is the text for Choice 2. It goes on and on and on and on and on and on. It can go multiple lines if it needs to. This is the text for Choice 2. It goes on and on and on and on and on and on. It can go multiple lines if

This is the text for Choice 3. It goes on and on and on and on and on and on. It can go multiple lines if it needs to. This is the text for Choice 3. It goes on and on and on and on and on and on. It can go multiple lines if it needs to. This is the text for Choice 3. It goes on and on and on and on and on and on. It can go multiple lines if

Above: This is a screenshot of what our Unity client looked like at this point, before it started loading events. It was very simple, just the event text, buttons for each choice, and the text for each choice.



The ghosts crowd closer in around you. Your teeth are chattering. They do not seem as though they are going to leave anytime soon. What do you do?

Plead with the ghosts to leave you alone.

Threaten the ghosts to leave you alone.

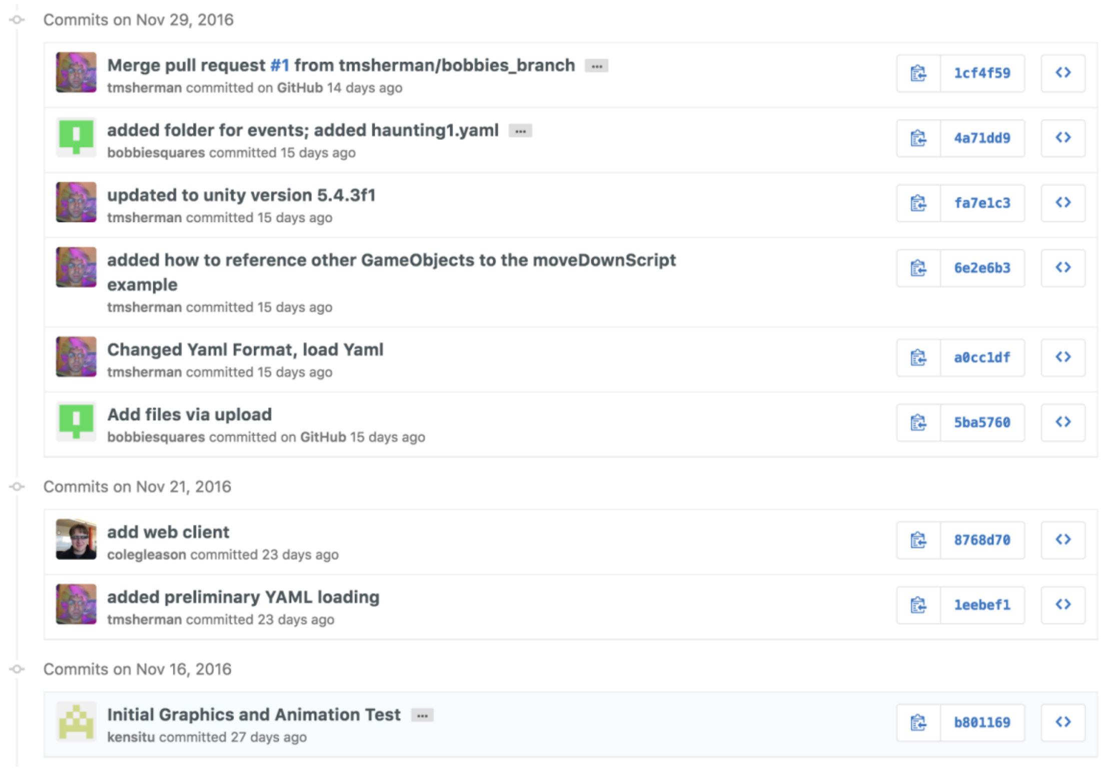
Try to use a spell you learned for banishing ghosts.

Ask the ghosts for advice.

poochz0rz: NES rivals ninja gaiden or some crap
poochz0rz: in terms of unforgiveness
poochz0rz: The missile have started to move!
poochz0rz: Did I miss something where you went to the beginning
poochz0rz: o no
poochz0rz: The grenade launcher have started to move
poochz0rz: The pizza is aggressiv.e
poochz0rz: NES games tend to not be very long right?
poochz0rz: What are you using to play this?
poochz0rz: Ah. I played around with a few. Can't remember why I didn't like a few of them but settled on VirtuaNES
poochz0rz: "Press Square to regenerate health. Can you go one more?"
poochz0rz: Did you fix the stream wrecker?
poochz0rz:
quasarinova: I'll see you later Thanks for stopping by
poochz0rz: NES rivals ninja gaiden or some crap
poochz0rz: in terms of unforgiveness
poochz0rz: The missile have started to move!
poochz0rz: Did I miss something where you went to the beginning
poochz0rz: o no
poochz0rz: The grenade launcher have started to move
poochz0rz: The pizza is aggressiv.e
poochz0rz: NES games tend to not be very long right?

Above: Our artist’s mockup of what a stream of The King and the Crowd (then called The King and the Court) might look like, before trying to implement these art assets in the Unity game. We made a few changes (such as to the icons used for the stats) and couldn’t include all the features shown here (such as minimizing stats, or viewing past choice history) in our proof-of-concept.

We used GitHub to store our game files, particularly the Unity game, and make them available to anyone in the group to download and edit. At this point, as we had multiple people working on creating resources and assets for the game, the Git history became quite busy. You can track our progress by looking at the commit timeline:


Commits on Nov 29, 2016

Merge pull request #1 from tmsherman/bobbies_branch
tmsherman committed on GitHub 14 days ago
1cf4f59

added folder for events; added haunting1.yaml
bobbiesquares committed 15 days ago
4a71dd9

updated to unity version 5.4.3f1
tmsherman committed 15 days ago
fa7e1c3

added how to reference other GameObjects to the moveDownScript example
tmsherman committed 15 days ago
6e2e6b3

Changed Yaml Format, load Yaml
tmsherman committed 15 days ago
a0cc1df

Add files via upload
bobbiesquares committed on GitHub 15 days ago
5ba5760

Commits on Nov 21, 2016

add web client
colegleason committed 23 days ago
8768d70

added preliminary YAML loading
tmsherman committed 23 days ago
1eebef1

Commits on Nov 16, 2016

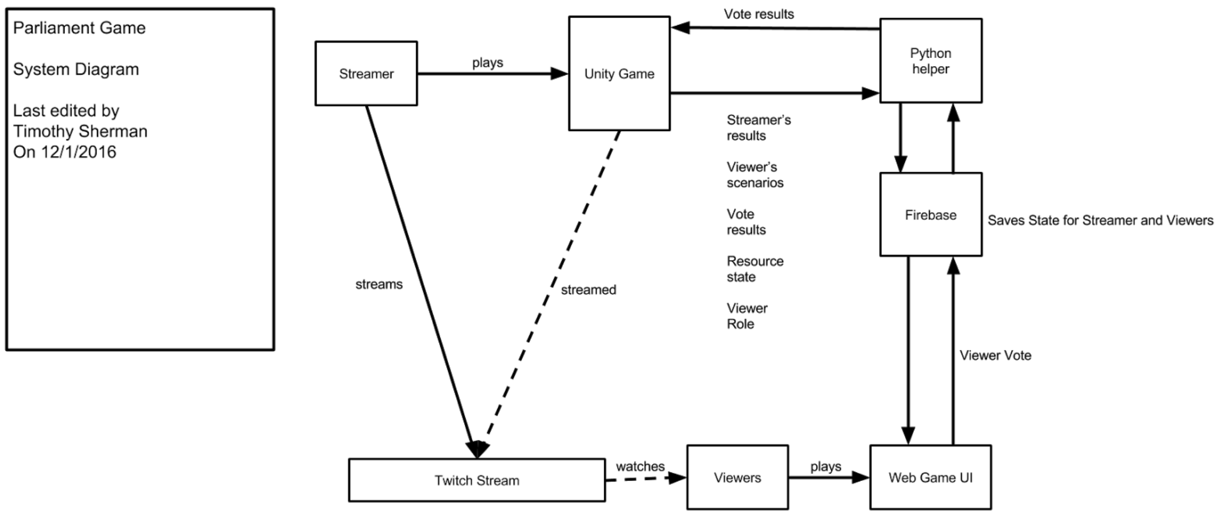
Initial Graphics and Animation Test
kensitu committed 27 days ago
b801169

Above: A screenshot of our Git project's commit timeline, which you can look at for yourself by following the link above. Here you can see art asset uploads, web client edits, event uploads, and Unity game edits.

Implementation Challenges

We quickly realized that creating events that were formatted appropriately, and had detailed information on the stats needed to both see the event and make different decisions, was very time consuming. Consequently, we ditched the idea of a paper playtest and focused on creating just enough events to illustrate the different kinds of events we wanted: events for each role (streamer, peasant, etc.), events with different outcomes, events with different stat checks, and events that triggered other events.

There were technical difficulties in the web client, such as figuring out how to record votes and display outcomes, and how to make the window resizeable. We were also still having difficulties with getting the different parts of the game--Unity, web client, and Firebase--to talk to each other. We ended up creating a helper application in Python to communicate between Unity and Firebase, that served much of the same purpose we originally envisioned the server having.



Above: The revised system diagram, showing what the system looks like currently. The Python helper acts as a medium between Firebase and the Unity game.

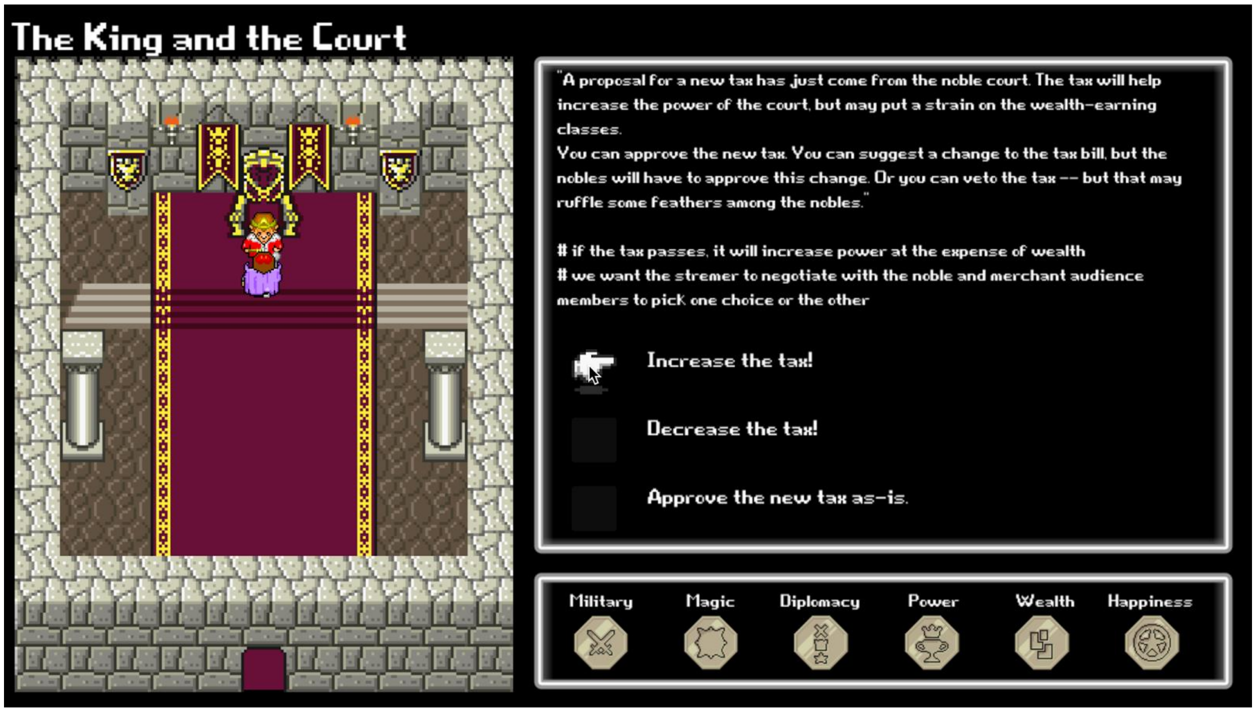
The Final Proof-of-Concept

With the deadline for the project fast approaching, we decided on creating a proof-of-concept rather than a balanced game. To that end, we had to give up many of our design ideas: with only ten working events, it's impossible to have the streamer last as long as possible, and there's no way the streamer will build up their stats high enough to actually pass the stat checks for most of the choices. There's also not enough events to procedurally generate the story, so for the proof of concept, we have a predetermined event order.

That said, we think this shows off the idea and concept for the flow of the game quite well. The events we have loading into the game cover a variety of events and concepts, including events for each class, and events that trigger other events across multiple classes. The events themselves are in the slightly silly fantasy setting we appreciated in our inspiration games, and have a variety of interesting choices and outcomes.

The current Unity game matches up with the concept screenshot mockup, complete with art and animations, although only one animation is used currently. The web client displays events, a choice, and Twitch chat, and a voting interface. It also greatly draws from our chosen concept sketch. And, most importantly, all the different parts are talking to each other! Our project is a successful proof-of-concept of this audience-participation narrative game system, and provides a framework for further development.

Below: A screenshot of the Unity game as it appears currently (save for a few minor changes, such as updating the name of the game to "The King and the Crowd"). The user can select from the choices displayed below the event text, and the resource stats can be shown by hovering over their respective icons.



The King and the Court

"A proposal for a new tax has just come from the noble court. The tax will help increase the power of the court, but may put a strain on the wealth-earning classes.
You can approve the new tax. You can suggest a change to the tax bill, but the nobles will have to approve this change. Or you can veto the tax -- but that may ruffle some feathers among the nobles."

if the tax passes, it will increase power at the expense of wealth
we want the stremer to negotiate with the noble and merchant audience members to pick one choice or the other

Increase the tax!

Decrease the tax!

Approve the new tax as-is.