Team Name: concurrent_butt_kicking
Members: Johanna Ross, Jacob Bennert, Will Rusk

The Project

Our project is a party game where you get to shape the story with your own personal flair. The players are given a series of prompts to fill in scenes in a story. They will then answer the prompt in whatever way they think fits the bill, and wait for the others to fill theirs in. Next, the scene will play out with each of the player's answers. The players can then vote on who's scene was the funniest/best and that player will receive points depending on how many votes they get. At the end of the game the story is told in full, and the players can relive the best entries.

EX:

As the heroes leapt out of the closet, the villain turned around and shouted in surprise, _____.

<Players 1-4 enter answers>

----- Player 1 -----

As the heroes leapt out of the closet, the villain turned around and shouted in surprise, "Please don't touch my butt!"

----- Player 2 -----

As the heroes leapt out of the closet, the villain turned around and shouted in surprise, "Welcome to McDonald's, can I please take your order."

----- Player 3 -----

As the heroes leapt out of the closet, the villain turned around and shouted in surprise, "I knew the coat people would return with a vengeance!".

----- Player 4 -----

As the heroes leapt out of the closet, the villain turned around and shouted in surprise, "EEK!", and clutches his pearls.

Vote for the best scene!

<Players 1-4 vote here>

Voting:

Player 1: [] [] Player 2: :(

Player 3: [] Player 4: []

Player 1 wins! The story continues with their submission inserted into the story.

<Next prompt>

What's the minimum/maximum deliverable?

Minimum deliverable:

- Text based prompts and answers
- One story, uncustomizable
- Support for 4 players

Maximum deliverable:

- Multiple pre-programmed stories
- The ability for players to create their own stories
- Text, drawing, sound and video prompts
- Text-to-speech recitation of the story once it's been completed with the players answers
- Visual effects / images with each scene
- Support for up to 8 players

What's your first step?

We will explore data structures and algorithms to represent our game and the game state. We need to think of how we will represent/store stories, and how that information will be distributed to the players.

What's the biggest problem you foresee or question you need to answer to get started?

Getting the producer/consumer relationship to work between PlayerHandlers and the Game Logic.

Here is a problem that we want to address:

How to communicate between machines

Here are two or three ways we considered addressing it:

- Using purely erlang
- Using python and a communication package
    - RabbitMQ
    - Socket Library
    - Ray

Here is the one we chose (at least provisionally) and why:

We ultimately decided upon the Socket Library because we felt it was the best suited to our purposes. We thought that Python would be easier to write a large program in than Erlang, even though Erlang does have very convenient message passing functionality built in. The Ray library, while very well-suited to our purposes, provided a little *too much*

functionality, and we felt as though using it would be letting the library do most of the work. RabbitMQ seemed more the kind of thing we were looking for, but the library was designed for very large programs, with hundreds of computers communicating amongst each other, which we decided was overkill for our needs. The socket library is already built in to Python, which is a plus, and upon researching the way the library was used, we thought it was fairly simple to grasp, and ultimately decided it would be the best decision.

Classes:

Server:

- FileHandler
    - Interfaces with data files and make data structures to represent story & story data
    - Stores results of game “best story so far” in output files for later replay
- InitializeHost
    - Initializes the game, e.g. lets the host start the game, lets the players join the game, then gives that info to the game logic and lets it run
- GameLogic
    - Implements game logic
    - Coordinates other classes, high level running the game

Client:

- ClientIO
    - Outputs user interface / render game state sent from server
    - Gets input (votes, scene info, etc) from player
- ClientLogic
    - Runs the client side of the game at a high level
- InitializePlayer
    - Creates a connection with the server running the game and then starts the client logic and lets it run

Network:

- PlayerHandler
    - Communicates with a single client instance, a list of these will share the game state to update client’s game state, request input, etc.
- ServerHandler
    - Class to communicate with server and send/receive relevant data

Files:

- createGame.py
    - This is the file that a user will run when they want to create a lobby for the game
- joinGame.py
    - This is the file the user will run when they want to join an existing game

Development Plan:

What Has Been Done So Far:

- createGame.py and joinGame.py have been completed
- InitializeHost and InitializePlayer classes have been completed
- Succeeded in sending a message over the network using Python's socket library
- Created bare bones user interface to display scenes and switch between pages
- GameLogic, ClientLogic, and FileHandler are all partially complete
- Syntax for storing the story in a file and reading it in has been developed and implemented

Distribution of Work:

- Johanna will be working on the client server side
- Jake will be working on the game server side
- Will will be working on the networking, i.e. the communication between server and client

Timeline:

- November 10: Client interacting with server in some way.
- November 17: Some basic functionality of client and server.
    - Bare bones user interface to show client scene and get user input
- November 24: Classes mostly finished. Basic outline of story.
    - Communication between classes
- Dec 1: Finish up storylines. Work on stretch goals.
    - At least 2 stories finished
    - Capability to load and switch between different story lines (allow user to select story from start page)
    - Work on cleaning up user interface

Abstraction and Language:

- Scene: part of the story that is displayed to the user with a section the user must fill in
- Submission: client's proposed entry into the story
- Vote: the submission a client likes most
- Result: scene with winning submission inserted into the story

Changes Made:

- Split the Initialize class up into separate InitializeHost and InitializePlayer classes
- Created the programs createGame.py and joinGame.py
- Decided on using PyGame for the client game interface
- Updated interface to most of the classes
- ClientIO class dramatically increased in size