Writeup Notes

# Grombcross

## Overview

A picross puzzle game made by myself and several Society of Play members themed around Society of Play original characters and inside jokes.

I organized a team of 8 to develop a picross puzzle game using WPF, C#, .NET 6.0, MVVM design principles. I worked on this game after work hours during my internship at the time, using the same tech stack and design patterns as my internship project.

## Puzzle Generation System

1. Each puzzle is generated from just a pair of bitmap images: one black+white (generation image) and one full-color (completed image).
2. The puzzle generation script parses the filename of these bitmap images for information such as puzzle name and puzzle id, and then iterates through the pixels of the generation image to fill a 2D boolean array (representing the puzzle’s solution state).
3. When a puzzle is selected by the user to play, this 2D boolean array is sent as an argument to the game view to generate a game board with the relevant line hints.

An aspect of the puzzle generation system I’m very proud of is allowing the developer or user to easily add, change, or remove puzzles without touching the codebase. The puzzles are generated at runtime using the bmp images in the puzzle data folders (which are intentionally exposed to the user in the distributed build folder).

## Save Data as JSON

The save data format is a dictionary, which is saved into as a json file. The key is the puzzle name (string) and value is the puzzle’s completion state (boolean).

I determined that a dictionary was the perfect data structure for saving game progress because it accommodates for adding, removing, or changing puzzles (either by the developer or by the user via the exposed puzzle data folders) without any issues.

* When loading, all the puzzles in the json file but not in the current game version will be ignored.
* When saving, only the puzzles in the current game version are included in the json file. So, previously-saved puzzles no longer in the game are wiped from the save data.

## UML Diagram

Created a UML to plan out game systems. This helps a lot when setting the project up! Removes a lot of guess work that would otherwise come with programming a project from scratch and keeps the scripts focused.

## Auto-Resizing Board

Game board auto-sizes to fit the window.

When the game view is loaded, the game board’s scale is calculated and set based on window height and width. While in-game, the board scale is recalculated and the board is re-rendered each time the window is resized via a function that listens for the window resized event.

# Project 9th Monday

## Overview

An audio-only horror narrative game following a group of kids on Halloween night looking to raid a haunted house for jewels to fund a video game console purchase.

Organized and led a team of 8 for a 10-day game jam. The team composed of two programmers, three voice actors, two writers, and an audio engineer. Game utilized the Wwise audio engine to handle audio clip and pitch randomization, and sequencing for ambience and dialog lines. Game ranked 15th out of 92 total entries.

## Determining Which Speech-to-Text API to Use

For this game, players interact with the world by speaking into a microphone. We wanted to have natural dialog in the game, so the player should’ve been able to speak as they would normally, and our systems would parse their speech for keywords to determine what action to take. In order to accomplish this, we looked into a few different speech-to-text APIs:

1. First, we looked into a variety of machine learning text-to-speech APIs. All of the ones we could find on the Unity asset store and elsewhere were paid assets, so we instantly ruled this out as an option (the spirit of a game jam involves making and publishing the games for free).
2. The second option we looked into was a built-in Windows API called Windows Keyword Recognizer. This API was free and fast, but had an issue that it could only detect keywords (rather than full sentences). For example, if one of our keywords is “blue” and a player speaks “blue car,” that phrase would be completely ignored. Because we are trying to simulate natural dialog in our game, we had to rule this option as well.
3. The third option we looked into was another built-in Windows API called Windows Dictation Recognizer. This API would register everything said by the player and send it to a script for our system to parse for keywords—exactly what we were looking for! The only downsides to this API were that the processing time was slower than keyword recognizer and it required the user to enable a speech recognition setting in Windows before playing, but we considered those acceptable drawbacks considering everything else worked perfectly!

## Organizing a Multi-Talented Team

I was team lead for this project, and there was a lot of communication I needed to have with all team members in order for us to complete such a strange game!

I had multiple meetings with the two writers to determine what type of story would be feasible to do in jam time and also what would be possible with our format of no visuals and listening for player speech. There were a lot of unique writing challenges with our format, such as making it obvious what choices the player has available at a given time and also making it clear when a player’s choice isn’t valid.

I had a few meetings with our audio engineer to teach him how to use the Wwise audio engine. I had previously used it for another game project, so I showed him how to import audio assets, apply effects, create sequences, and expose audio events for invocation in Unity scripts.

I created an asset list and a gantt chart for the team to clearly communicate assets required and the production timeline. This was actively followed and updated by team members, and it proved useful as our game was completed on-time.

# Pizza Delivery Bagel

## Overview

A chaotic action-arcade game about delivering as many pizzas as possible before inevitably getting fired.

I developed this game entirely by myself (including programming, game design, art, sound effects, music) in my free time over 1.5 years. I used this game as an opportunity to learn various things: implementing game audio through the Wwise audio engine, mobile game development, cross-platform development, cloud saving, tools development, and seeing a project through to completion.

The game took as long as it did as a result of me being a full-time student and development overlapping with two summer internships and several game jams.

## Simultaneous Android and PC Development (Support different controllers and different playstyles)

Pizza Delivery Bagel supports multiple different control schemes: keyboard, touch, gamepad, and arcade cabinet (for the Dallas Society of Play arcade cabinet version!). When starting a game, the control scheme is set based on the detected platform.

If playing on a PC, the keyboard and gamepad control schemes can be switched-between mid-game.

Depending on the control scheme, some control adjustments are made to improve game feel.

* Example: Wall grinding is made easier when playing with a joystick. This is because the wall-grinding speed boost is most effective when moving into a wall at a 45 degree angle, which is easiest to do on keyboard directional arrows.
* Example: When playing on a PC with a gamepad or an arcade cabinet, car acceleration and deacceleration are mapped to separate buttons rather than to the up and down directions. I felt this made the gamepad controls feel significantly more natural.
* Example: When playing on mobile, the car moves in the direction of the virtual joystick, rather than using tank controls. I found that the tank controls felt too awkward and unforgiving on a touchscreen.

## Object Pooling

Utilized object pooling to minimize garbage collection and improve performance. Nothing is destroyed or creating while playing the game. Instead, all objects are created at the start of the game and disabled, enabled, and re-positioned over time as needed!

## Development Tools

I created some custom inspectors with debugging shortcuts.

In-game terminal for debug commands. I used a 3rd party terminal because it provided all the utility I needed and saved time.

## Presentation Skills

Presented progress updates regularly during weekly discord game showcase events. Presented game at two local game conventions for playtesting and feedback.

# The Eaterer

## Overview

A first-person horror game about avoiding unhealthy lifestyle choices. The player is chased throughout a series of pizza box mazes by a fat blob known as The Eaterer. If the player isn’t able to make the correct lifestyle choices, they become an “Eaterer” themselves…

I developed this game during the start of quarantine in 2020 with a classmate. This was my first Unity 3D game and my first non-solo programming project. We originally planned to publish the game onto the Steam marketplace, but design disagreements between us cut the development short. We were able to compromise on a scaled-back idea for the game and eventually released it for free on Itch.io. This project taught me a lot about game development, programming, and collaborating with a team.

## Reading Items from Excel Spreadsheet

Item data was stored in an excel spreadsheet.

In the spreadsheet, items had two columns of data: a column with their name, another column with a Boolean if they’re good/not. From a C# script, this spreadsheet was read and used to generate items throughout the map. The 3D models were stored in a separate, public array variable of the script (but, in hindsight the models should’ve been referenced in the spreadsheet in some way).