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 for a game jam. 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 filenames of these bitmap images for information—such as puzzle name and id number—and then iterates through the pixels of the black+white generation image to create 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 player to easily add, change, or remove puzzles without touching the codebase. The puzzles are generated at runtime using the .bmp images in the exposed puzzle data folders. Puzzles can be added, changed, or removed by modifying their respective .bmp image pairs (which requires only a basic image editor).

## Save Data as JSON

The save data is formatted as a dictionary and saved into a json file. The dictionary’s key is the puzzle name (string) and its 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 touching the codebase.

* When loading, any puzzle save data found in the json file without a matching puzzle in the puzzle data folder will be ignored.
* When saving, only the puzzles currently in the puzzle data folder are saved into the json file. So, previously-saved puzzles that are no longer in the game are excluded.

## UML Diagram

I created a UML to plan out game systems. This helped a lot!

I enjoy creating UMLs before starting to program because I’ve found it removes a lot of guess work that would otherwise come with programming a project from scratch and helps keeps scripts focused. I’ve been burned too many times in the past by cowboy coding and then discovering months down the line that my projects are unsustainable because of a messy codebase...

## 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 re-calculated 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+pitch randomization and to handle sequencing for ambience tracks and dialog lines. Our 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: the player would speak as they would normally and our systems would parse their speech for keywords to determine what action to take. To accomplish this, we looked into a few different speech-to-text APIs:

1. First, we researched a variety of machine learning speech-to-text 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 because the spirit of a game jam contest involves developing and publishing games entirely for free.
2. The second option we looked at was a built-in Windows API called Windows Keyword Recognizer. This API was free and incredibly fast, but had a deal-breaking issue that it could only detect keywords (rather than full sentences). For example, if one of our keywords was “blue” and a player spoke the phrase “blue car,” everything the player said would have been ignored. Because we are trying to simulate natural dialog in our game, we had to rule this option out 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 the Windows Keyword Recognizer and the use of it required the user to enable a speech recognition setting in Windows; but, we considered those acceptable drawbacks considering everything else worked perfectly and decided to use it!

## Organizing a Multi-Talented Team

I was the team lead for this project and there was a lot of communication and organization required from me for us to complete such a unique game within only 10 days!

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 given our unconventional game format of no visuals and player interaction done through speech. There were a lot of unique writing challenges with our format, such as making it obvious what choices the player had available at a given time and also making it clear when a player’s choice isn’t valid. As the one who knew the limitations of our technology, I had to constantly communicate with the writers and review their work to ensure the interactions would be possible with our format.

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). I showed him how to import audio assets, apply effects, create sequences, and expose audio events for invocation in Unity scripts. The audio in our game was incredibly important since it was the sole source of feedback the player got from game world, and everything audio-related came out great!

I created an asset list and a gantt chart for the team to clearly communicate the assets required from each team member 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 (programming, game design, art, sound effects, music, etc.) 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 the game’s development overlapping with two summer internships and a few 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. When starting the game, the control scheme is set based on the detected platform.

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

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

* Ex. When playing with a physical or virtual joystick: wall grinding detection is more sensitive. 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 reliably do with keyboard directional arrows.
* Ex. When playing on a PC with a gamepad: car acceleration and deacceleration are mapped to the triggers rather than to the up and down movement directions. On keyboard it’s fun to control the car entirely with the arrow keys, but on a joystick or d-pad it doesn’t feel natural.
* Ex. When playing on mobile: tank controls aren’t an option in the settings. The tank controls feel too awkward and unforgiving on a touchscreen, so they’re made exclusive to the PC and arcade cabinet versions.

## Object Pooling

I 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.

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

## Presentation Skills

I 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 action game about avoiding unhealthy lifestyle choices. The player is chased by a fat blob—known as The Eaterer—through a series of pizza box mazes. 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. Us two had originally planned to publish the game onto the Steam marketplace, but design disagreements between us eventually 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 in a team.

## Reading Items from Excel Spreadsheet

Item data is 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 (in hindsight it would have been better to reference the models in the spreadsheet in some way).

# Monkey Man Enclosure

## Overview

An action arcade game about a man who has sleep-walked into a monkey enclosure (wearing a monkey costume). The junior zookeeper must keep the man safe until he wakes up.

I organized a team of 5 to develop this game for a 10-day game jam. The team included: a 3D artist, a UI artist, a composer, and two programmers. Our game ranked 3rd out of 22 entries!

## Object Pooling

To minimize garbage creation and collection while playing (to avoid dropping frames due to de-allocating memory), throwable objects are pooled rather than being created and destroyed. There can only be at most 6 bananas, 6 bricks, and 30 food pellets on the ground at one time.

To mask the re-use of throwable objects, the objects subtly scale out of existence shortly after colliding with the ground.

## Project Management

Before my co-programmer and I started touching the code, we had a meeting to map out the game systems and create a UML diagram. This simplified our development of systems a lot and helped clarify which systems interact with which. Also, it made it easier to judge complexity of systems to divide-up tasks between the two of us.

My co-programmer and I used Github as our source control. We worked on features in separate feature branches and merged these into main once we completed our work.

I created a collaborative asset list for all of the group members and regularly checked-in with all of them to make sure everybody knew what to be doing.

# Continuous Glucose Monitor Nightlight

## Overview

Some background: I’m a type 1 diabetic and I use a CGM (continuous glucose monitor) to monitor my blood sugar throughout the day and at night. My CGM plays an alarm on my phone whenever my blood sugar is critically high or low. At night, it’s especially important that I hear this alarm because an untreated low blood sugar can be incredibly dangerous. However, I have bad hearing! I often sleep through this alarm at night when it goes off.

So, I decided to set up a smart lamp to shine a light at me at night in place of the alert tone to hopefully wake me up more reliably.

The following is a diary of my experiences with this ongoing project: problems I’ve ran into and how I solved them.

### Setup

To start, I researched if any others online had done similar projects. I was only able to find one person who had done something similar, and luckily they published a tutorial article. It wasn’t a particularly detailed tutorial, but it helped a lot! It pointed me in the right direction regarding what technologies to use and proved to me that the project was feasible.

<https://libbyrome.com/health/type-1-diabetes/flashing-my-lights-when-my-blood-sugar-is-too-high-or-too-low/>

I decided to use the following tech stack recommended by the article:

* The web app Nightscout to gather information from my CGM (Dexcom G6) via the Dexcom API.
* Heroku to host the web app.
* MongoDB to store my blood sugar readings.
* IFTTT to listen for Nightscout low+high blood sugar events and then invoke a smart light to turn on.

### Smart Lamp

Next, I had to find a smart lamp that was both compatible with IFTTT and bright enough to wake me up. After a few purchases+returns, I found a nice one!



## The Pi

I installed Raspberry Pi OS onto my Raspberry Pi 3 and set up the Nightscout web app on it. I edited the Pi’s startup script so that it opens the web app in as soon as the device turns on.

I initially set this up with a screen so I could see at-a-glance if the app was running (and because I thought it looked cool), but later I decided to remove the screen because it was too bright and I noticed I never found a need to look at it since my phone displays the same information.

A picture containing text

Description automatically generated

### Issue #1: I forget to turn my Pi on or off sometimes…

Because I didn’t want my lamp turning on in the middle of the day in the event of a critical blood sugar reading, I would turn my Pi on at night and off in the morning.

After several months, I realized I had trouble making this a routine and I would often forget to turn it on or off. So, my solution was to leave the Pi running continuously but only allow it to turn on my smart light within set hours (11pm – 7am). I was forced to upgrade to a subscription tier of IFTTT, but I felt it was a worthwhile price for this peace of mind.

Graphical user interface, text

Description automatically generated

### Issue #2: The lamp doesn’t wake me up sometimes…

After a few months I noticed that the smart lamp, regardless of light strength, was small enough that I could ignore it sometimes by turning over in my bed.

So, I figured it would be better to instead turn my room’s lights on. I bought and am currently using LIFX smart bulbs and have never once slept through the lights in my room turning off.



# Portfolio Site

## Overview

Motivated by a combination of wanting to learn Microsoft Blazor, wanting to program my own portfolio site from scratch, and wanting to stop paying Weebly for my old website, I decided to create this portfolio site!

I’m very happy with how it turned out! I programmed this all using HTML and C# (MS Blazor), utilized a UI component library Blazorise to make everything look nice, and hosted the site on Netlify.

## Using Blazor

Microsoft Blazor has been very fun to learn! For those that don’t know, the quirk with Blazor is that all the logic programming can be written in C# and executed natively on the .NET runtime. As someone who doesn’t enjoy javascript programming and has a lot of experience with C# development, I love it 😁

Because of how new Blazor is, there aren’t very many resources online like forum posts or how-to articles. For the most part, I was stuck with Microsoft’s and Blazorise’s documentation. This made debugging niche issues very annoying. Luckily, Blazor does also support javascript logic programming as a fallback! For difficult issues it’s possible to ignore C# and .NET and solve your issues using javascript resources.

## Animations

I utilized animations built into the Blazorise UI component library across my entire site!

For buttons  
For switching pages

If you’ve noticed, the page switching in my site is directional. The page change animation will play a “slide left” or a “slide right” depending on which direction the destination tab is from the current tab. When a page tab is clicked, a function is called that will set the direction of the animation based on what page the user is currently on.



## Dark Mode

Dark mode switching luckily wasn’t too tricky to implement using the Blazorise UI component library.

I created custom themes for light and dark mode and utilized a Blazorise theme-switching system for toggling the theme at runtime. Some Blazorise components—such as Cards—didn’t seem to be affected by theme colors for some reason, so I had to manually adjust their css properties for them to update correctly. I also added some unique functionality to the page tabs for them to share the color of their respective pages (different colors depending on if dark mode was on/off).

I wanted the dark mode state of the website to be auto-set to the browser’s dark mode state (light mode users see the light mode site by default, dark mode users see the dark mode site by default). Sadly, this issue was unsolvable using C# and ASP.NET… I had to implement a small javascript function to detect the browser’s dark mode state and then call that from a C# method as soon as the site loads.