Rush Hour

Unit: Critical Play and Development (PU002626)

SKYLAR LAW - 2204562

Rush Hour is a Slice of Life pixel-art game about grocery shopping during rush hour.

CONTENTS

1. Project Introduction

1.1. Personal Aims

- **Develop and design** a Playdate console game with a 1-bit visual style limitation, providing an opportunity for explore development for other platforms
- **Learn and apply new** techniques and **programming languages** to suit the project's needs.
- **Construct a well-structured narrative** that revolves around **slice-of-life elements**, seamlessly integrating them into the intricate game mechanics.
- **Implement multiple branching endings**, where the consequences of player's actions influences the **outcomes of the game**.

1.2. Initial Ideas

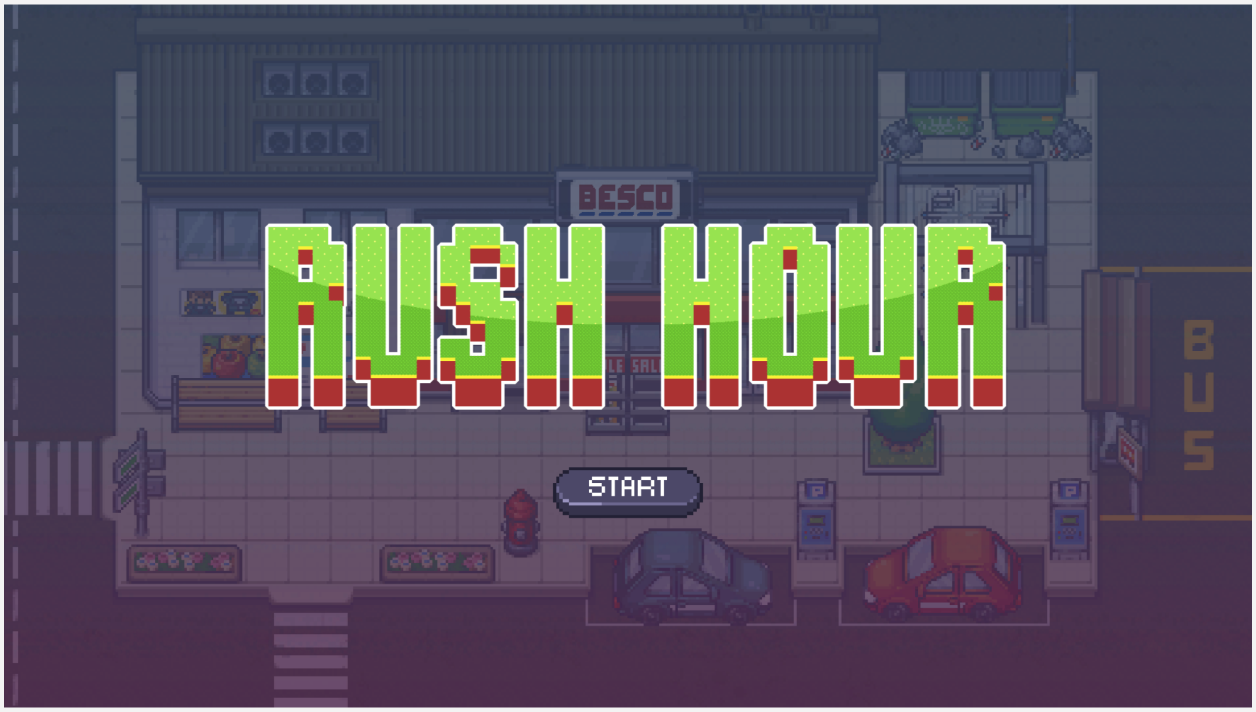
When it comes to brainstorming ideas for my game projects, numerous initial concepts would normally flood my mind. However, when I immersed myself in this topic of slice of life experiences, one particular scenario instantly captivated my imagination: grocery shopping. This routine activity is a consistent part of our lives, but it is during the frenzied rush hour moments that truly becomes a source of stress and annoyance.

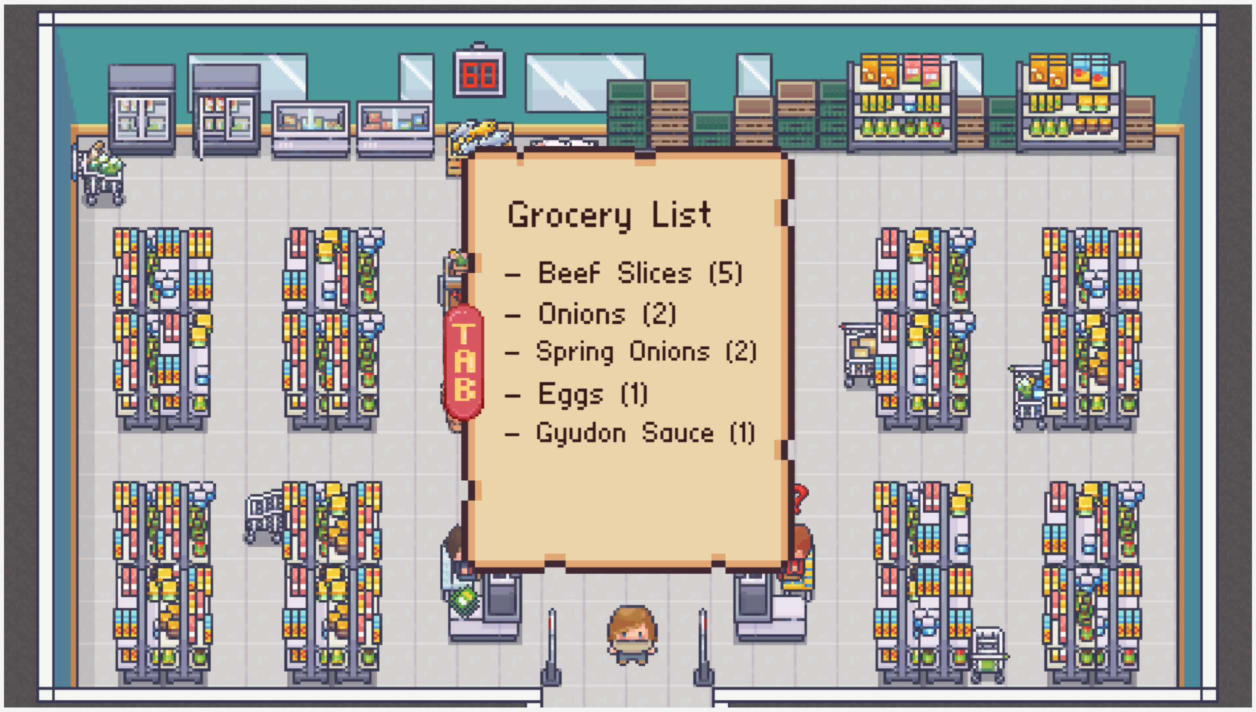
This game also serves as an autobiographical exploration of my everyday life, capturing the essence of my response to urgent situations. Personally, I tend to experience stress quite easily and possess a strong sense of urgency, always driven to accomplish tasks efficiently. Through this game, players will have the opportunity to step into my shoes and truly grasp the emotions and sensations that accompany these experiences. It aims to provide an authentic and relatable representation of what it feels like to navigate life with a constant need for productivity and accomplishments.

1.3. Developer's Note

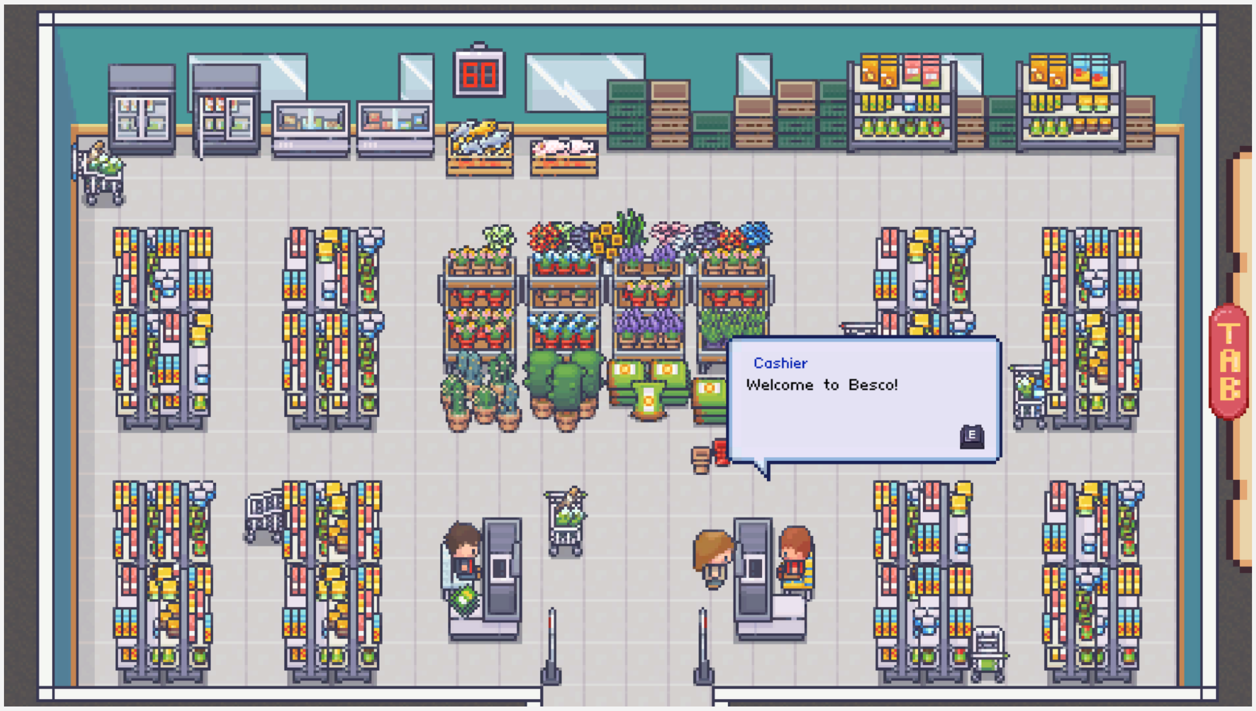
Although some of my aims were not met, the game and the development decisions were eventually modified to suit the timeline given for this project. This documentation will showcase all of the progress from the beginning (Pulp Editor version) to the end (Unity version). This documentation will also explain why the initial ideas and development tools were not feasible, seen in Development Changes.

1.4. Game Screenshots (Complete Version)

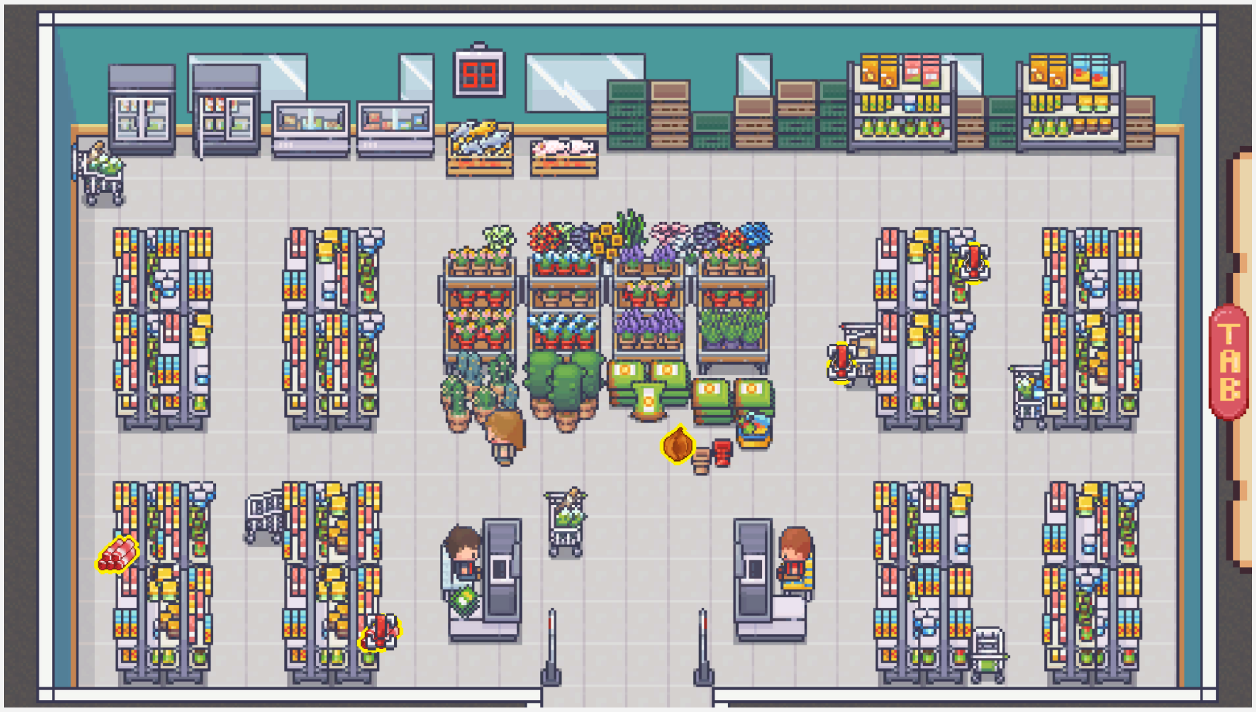




Grocery List
- Beef Slices (5)
- Onions (2)
- Spring Onions (2)
- Eggs (1)
- Gyudon Sauce (1)



Cashier
Welcome to Besco!



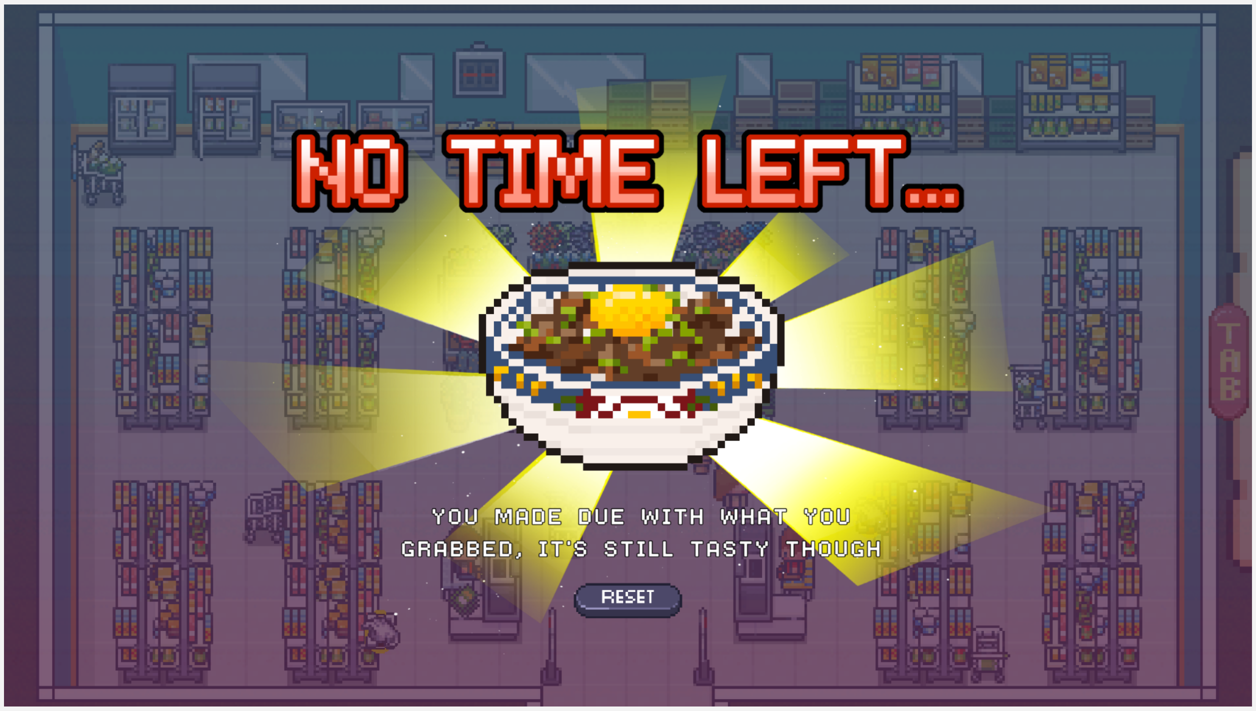


Figure 1-5: Screenshots of Rush Hour, the game

1.5. Development Tools

Tools Used	Application of Tools
Playdate Pulp Editor	Bitsy-inspired browser based game maker, used to create the original Rush Hour prototype in Game Jam #1.
Visual Studio Code	IDE used for scripting in Lua for the Playdate SDK.
Playdate SDK (Version 1.13.4)	The Software Development Kit to be
Unity (Version 2021.3.6f1)	The game engine which was eventually used to create the final game.
Visual Studio Community 2022	IDE used for scripting in C# for the Unity version of the game.
Miro	Used to manage game layout, systems and visuals. Overall game planning.

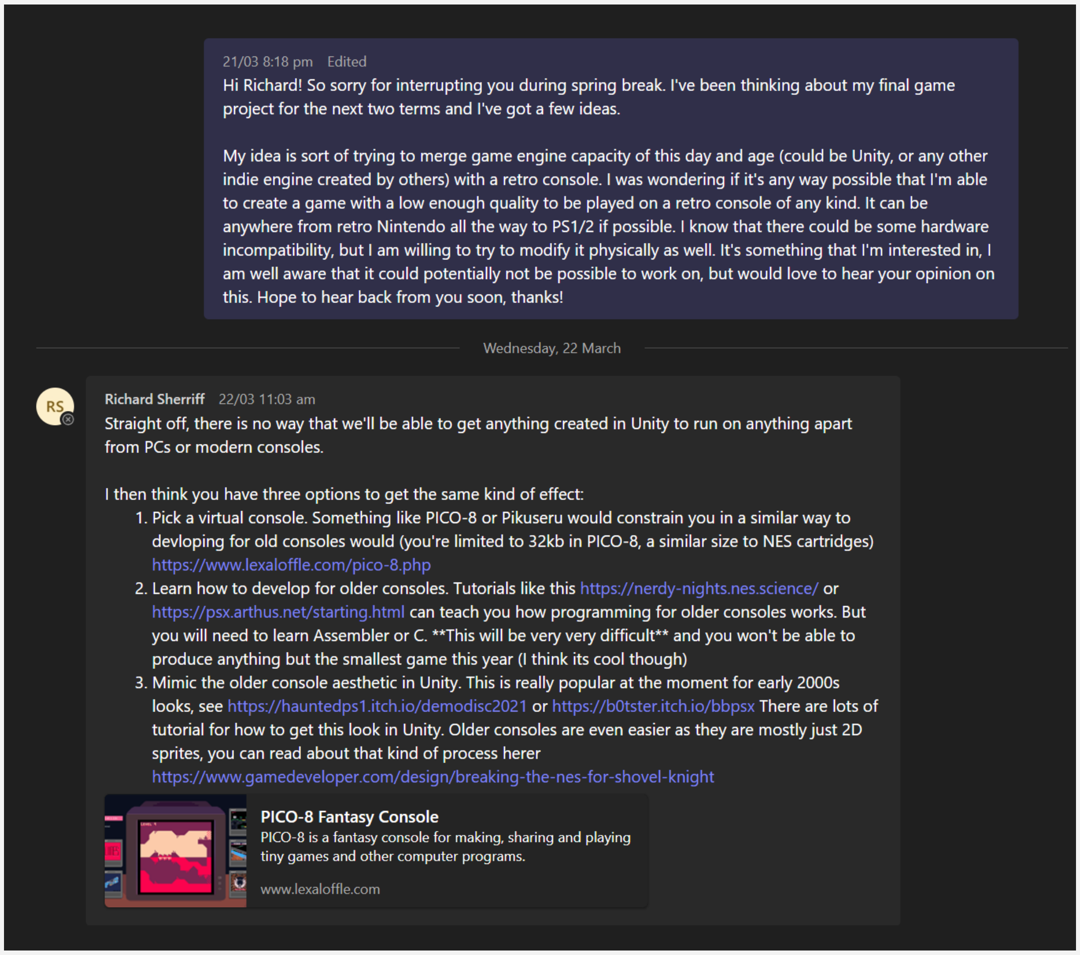
Google Docs & Microsoft Powerpoint	*Microsoft Powerpoint:* Used to create presentations for game showcases in Term 3, Week 6. *Google Docs:* Used to document all of the process of the game's development through this development log
Notebook (Physical)	Initial concepts, sketches and game designs were all drawn on paper. All images will be available here.

2. Technology

2.1. Initial Ideas & Choice

As I began brainstorming ideas for this project, I approached it as a preliminary exercise for my final project. Initially, I had a faint concept for my final project, aiming to develop a retro-style game, potentially designed for retro gaming consoles. The selection of this particular technological platform was driven by my deep appreciation for retro game development. Throughout the years, numerous game developers such as Puppet Combo and Chilla's Art, have evoked a sense of nostalgia by drawing inspiration from the visual styles of the PSX era. Rather than solely applying a PSX shader, my aspiration was to authentically recreate the immersive experience of playing a game from the 1990s by employing the actual technologies prevalent during that time.

In order to assess the feasibility of my vision, I sought guidance from Richard, seeking his expertise and insights. Richard provided several potential solutions to achieve the desired effect, one of which involved developing on the PICO-8. However, during my in-depth exploration of fantasy consoles, I stumbled upon the Playdate console. This compact handheld device possesses hardware limitations akin to the Gameboy, supplemented with a unique crank mechanism. I firmly believed that the Playdate console presented an ideal compromise for my project, with its inherent limitations in aesthetics and technology aligning seamlessly with my creative intentions.



21/03 8:18 pm Edited

Hi Richard! So sorry for interrupting you during spring break. I've been thinking about my final game project for the next two terms and I've got a few ideas.

My idea is sort of trying to merge game engine capacity of this day and age (could be Unity, or any other indie engine created by others) with a retro console. I was wondering if it's any way possible that I'm able to create a game with a low enough quality to be played on a retro console of any kind. It can be anywhere from retro Nintendo all the way to PS1/2 if possible. I know that there could be some hardware incompatibility, but I am willing to try to modify it physically as well. It's something that I'm interested in, I am well aware that it could potentially not be possible to work on, but would love to hear your opinion on this. Hope to hear back from you soon, thanks!

Wednesday, 22 March

Richard Sherriff 22/03 11:03 am

Straight off, there is no way that we'll be able to get anything created in Unity to run on anything apart from PCs or modern consoles.

I then think you have three options to get the same kind of effect:

1. Pick a virtual console. Something like PICO-8 or Pikuseru would constrain you in a similar way to devloping for old consoles would (you're limited to 32kb in PICO-8, a similar size to NES cartridges) https://www.lexaloffle.com/pico-8.php
2. Learn how to develop for older consoles. Tutorials like this https://nerdy-nights.nes.science/ or https://psx.arthus.net/starting.html can teach you how programming for older consoles works. But you will need to learn Assembler or C. **This will be very very difficult** and you won't be able to produce anything but the smallest game this year (I think its cool though)
3. Mimic the older console aesthetic in Unity. This is really popular at the moment for early 2000s looks, see https://hauntedps1.itch.io/demodisc2021 or https://b0tster.itch.io/bbpsx There are lots of tutorial for how to get this look in Unity. Older consoles are even easier as they are mostly just 2D sprites, you can read about that kind of process herer https://www.gamedeveloper.com/design/breaking-the-nes-for-shovel-knight

PICO-8 Fantasy Console

PICO-8 is a fantasy console for making, sharing and playing tiny games and other computer programs.

www.lexaloffle.com

Figure 6: Screenshot of conversation with Richard (with Richard's permission to share - asked on 9/6/23)

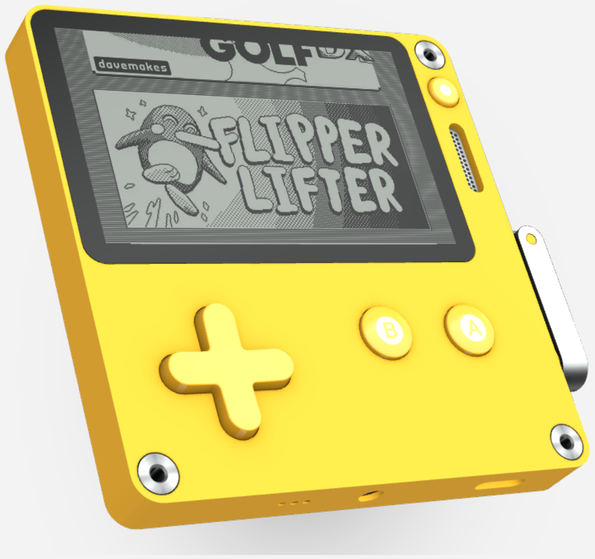
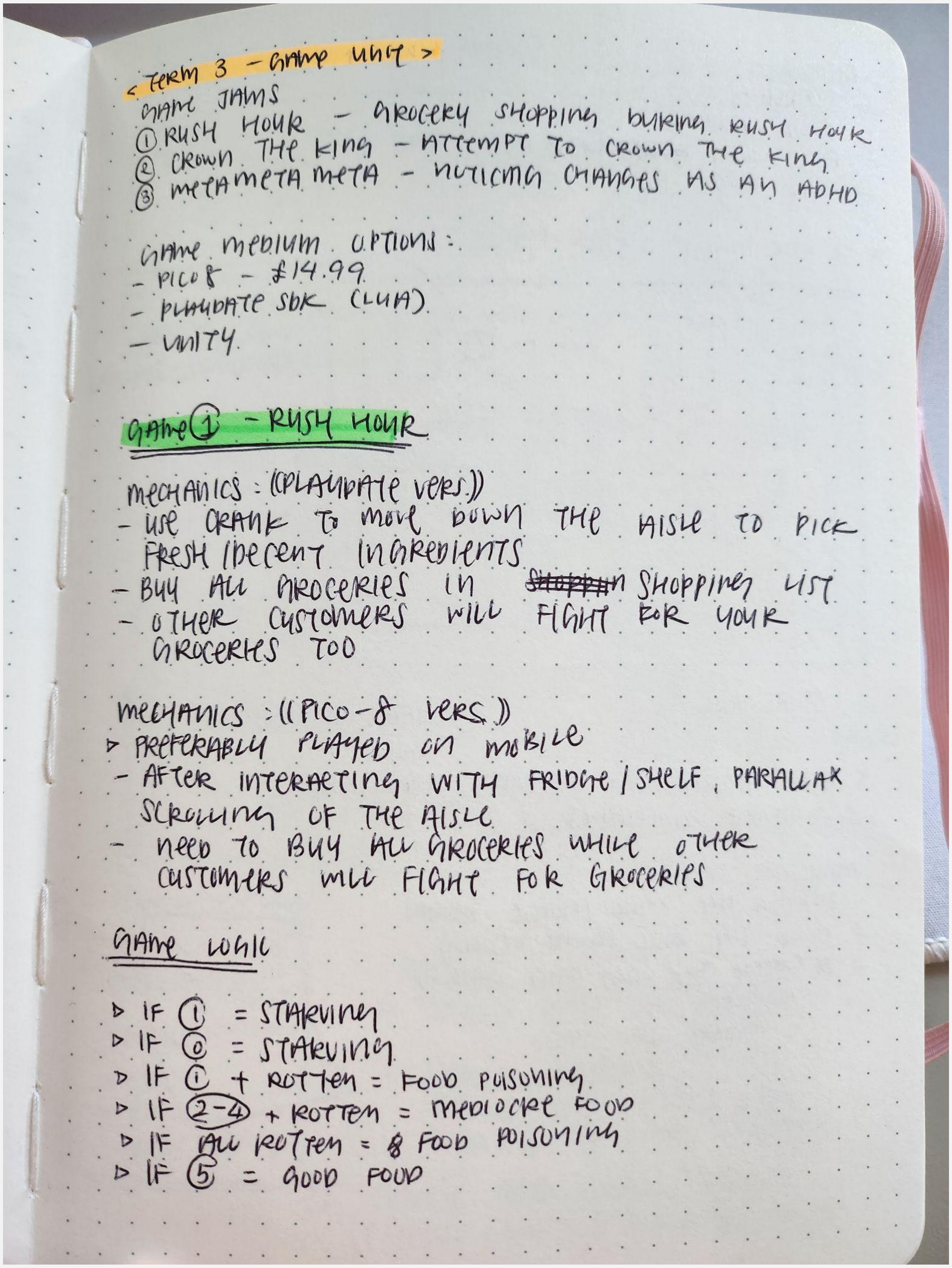


Figure 7: Playdate Console



< TERM 3 - GAME UNIT >

GAME JAMS
① RUSH HOUR - GROCERY SHOPPING DURING RUSH HOUR
② CROWN THE KING - ATTEMPT TO CROWN THE KING
③ METAMETA META - NOTICING CHANGES AS AN ADHD

GAME MEDIUM OPTIONS:
- PICO8 - £14.99
- PLAYDATE SDK (LUA)
- UNITY

GAME ① - RUSH HOUR

MECHANICS: ((PLAYDATE VERS.))
- USE CRANK TO MOVE DOWN THE AISLE TO PICK FRESH/DECENT INGREDIENTS
- BUY ALL GROCERIES IN ~~SHOPPIN~~ SHOPPING LIST
- OTHER CUSTOMERS WILL FIGHT FOR YOUR GROCERIES TOO

MECHANICS: ((PICO-8 VERS.))
▷ PREFERABLY PLAYED ON MOBILE
- AFTER INTERACTING WITH FRIDGE/SHELF, PARALLAX SCROLLING OF THE AISLE
- NEED TO BUY ALL GROCERIES WHILE OTHER CUSTOMERS WILL FIGHT FOR GROCERIES

GAME LOGIC

▷ IF ① = STARVING
▷ IF ⓪ = STARVING
▷ IF ① + ROTTEN = FOOD POISONING
▷ IF (2-4) + ROTTEN = MEDIOCRE FOOD
▷ IF ALL ROTTEN = 8 FOOD POISONING
▷ IF ⑤ = GOOD FOOD

Figure 8: Image of notebook of initial ideas jotdown

2.2. Research

In order to develop for the Playdate console, the two primary programming languages used to develop for a Playdate game were Lua and C. After careful consideration, I deemed Lua to be the most suitable choice, given its relative simplicity and my proficiency in handling it within the constrained time frame.

During my investigation into Playdate development, I came across an intriguing discovery: the creators of the Playdate console had developed a user-friendly browser-based game maker called Pulp Editor, drawing inspiration from the popular Bitsy framework. This tool empowered individuals with limited prior knowledge to engage in game creation effortlessly. Although extensive coding proficiency was not mandatory, certain mechanics necessitated the implementation of writing custom scripts. Thankfully, the creator of the Pulp Editor has provided a comprehensive and well-documented resource in the form of PulpScript documentation, accessible here.

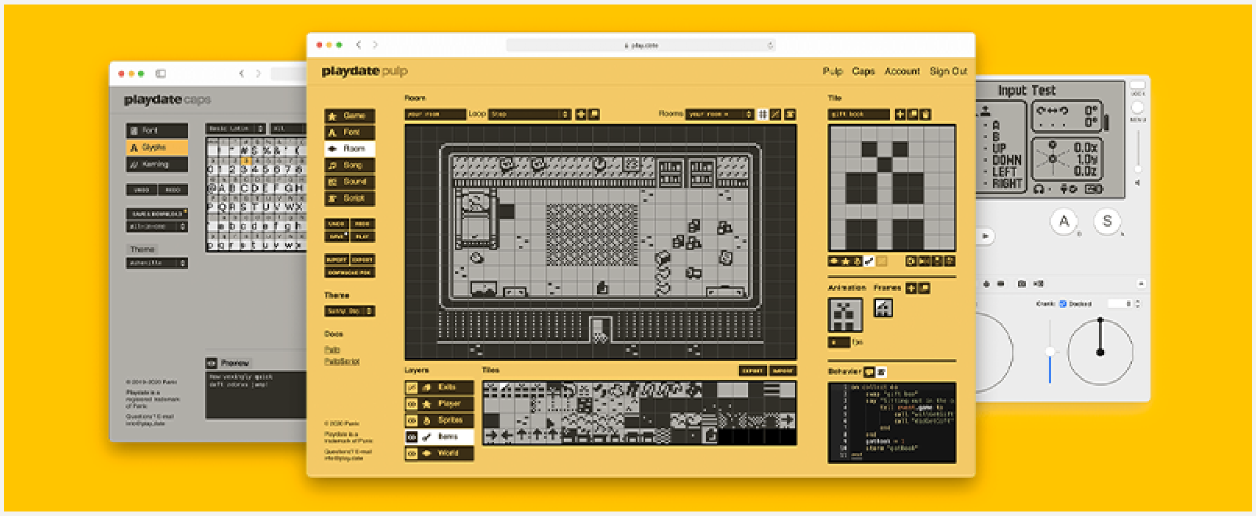
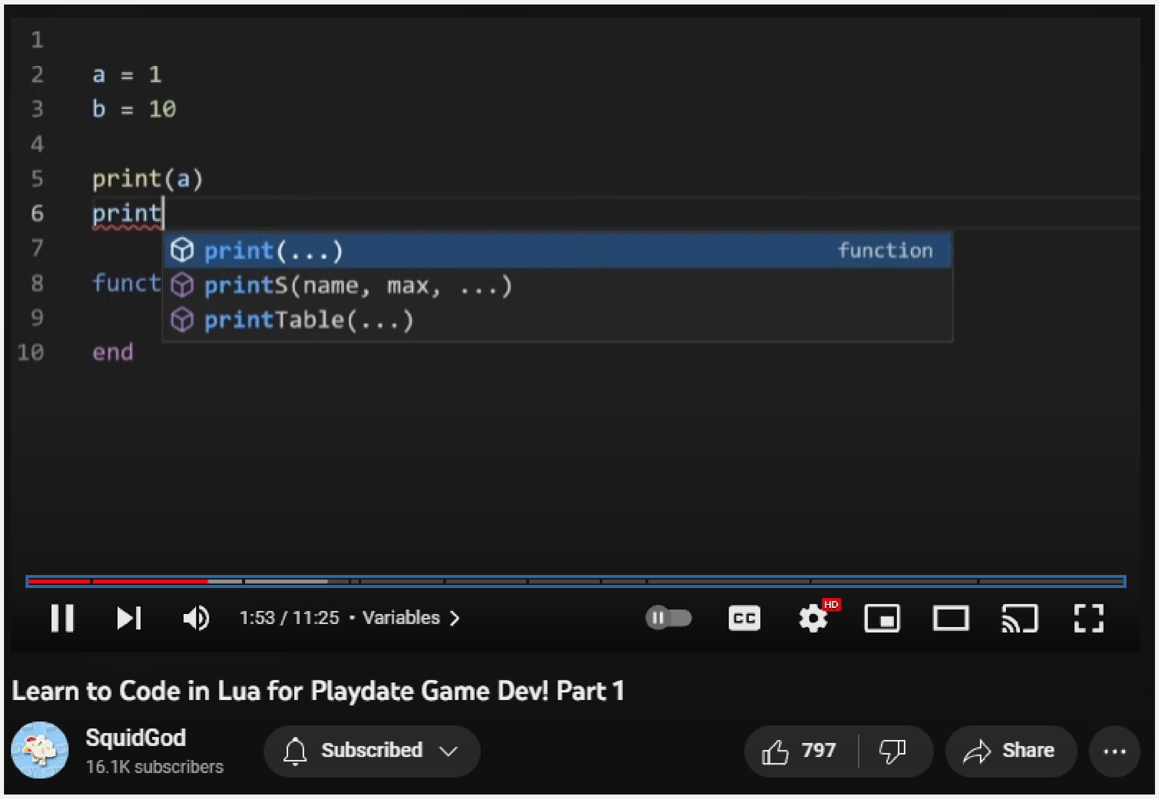
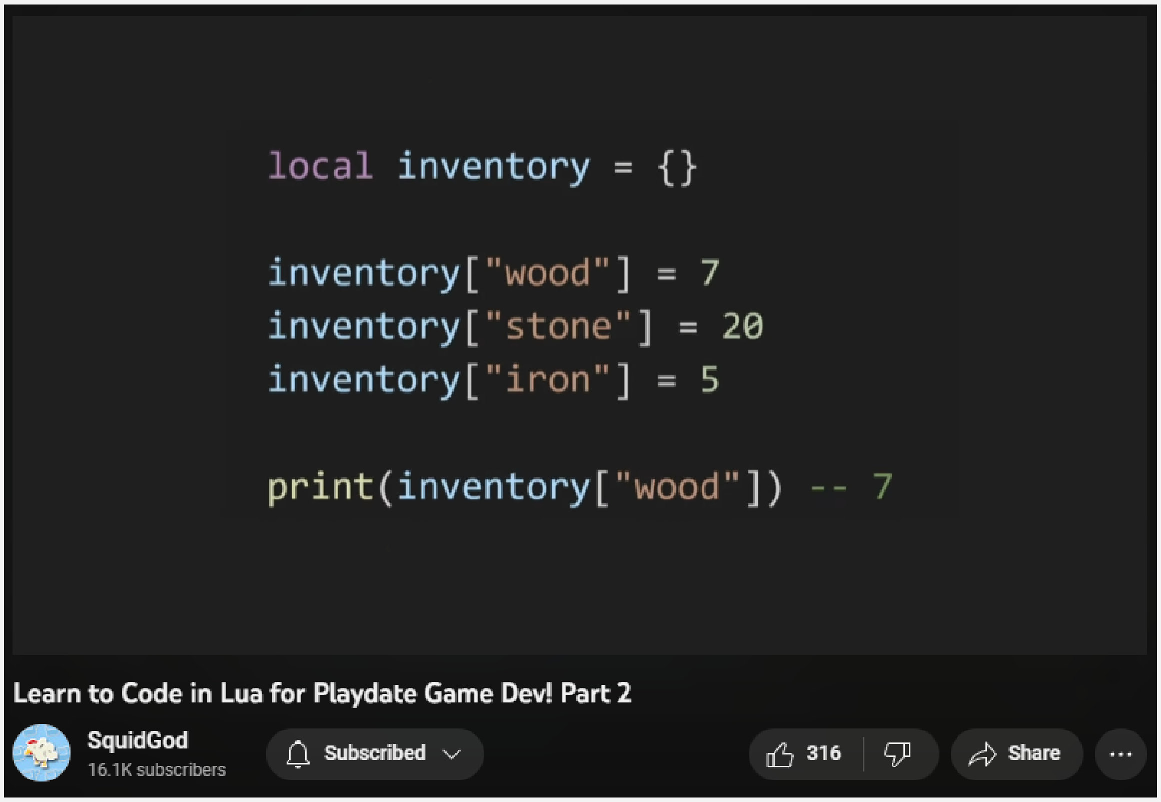


Figure 9: Playdate Pulp web browser features

Following my exploration of the Playdate prototype for the initial game jam iteration, I started an in-depth study of Lua to equip myself for the proceeding development process. To facilitate my research, I delved into a variety of tutorial videos and GitHub resources provided by SquidGod, an accomplished indie game developer who dedicates his free time to creating games for the Playdate platform. SquidGod has graciously shared a wealth of insightful videos tailored specifically for beginners seeking to acquire coding skills for the Playdate.



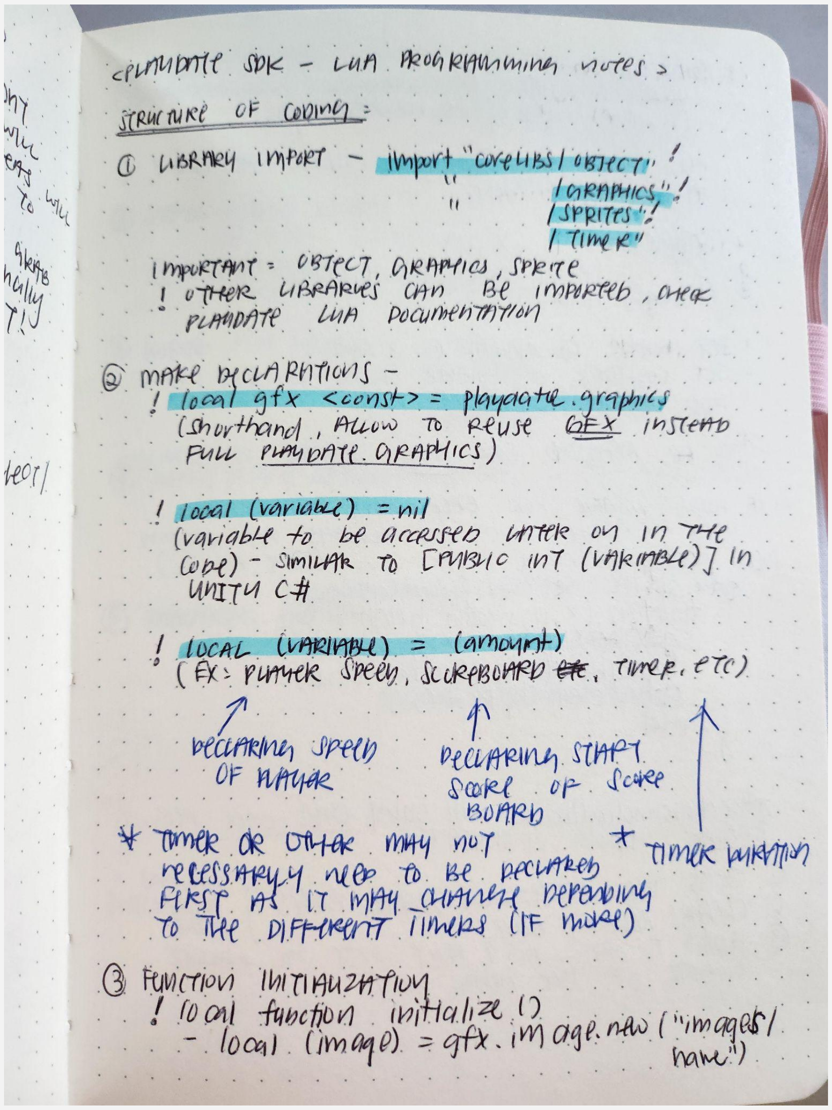
```
a = 1
b = 10

print(a)
print

funct
end
```



```
local inventory = {}

inventory["wood"] = 7
inventory["stone"] = 20
inventory["iron"] = 5

print(inventory["wood"]) -- 7
```

Figure 10-11: Screenshot of tutorial videos by SquidGod on YouTube

Gaining a thorough understanding of the underlying structure of the Lua programming language became a priority, although it may be tough, there are subtle similarities from C# and Unity's file management system. Consequently, I made a deliberate effort to document every piece of knowledge acquired during this extensive learning process.



<PLAYDATE SDK - LUA PROGRAMMING notes>

STRUCTURE OF CODING:

① LIBRARY IMPORT - import "coreLIBS/OBJECT" !
" /GRAPHICS" !
" /SPRITES" !
/TIMER"

IMPORTANT = OBJECT, GRAPHICS, SPRITE
! OTHER LIBRARIES CAN BE IMPORTED, CHECK PLAYDATE LUA DOCUMENTATION

② MAKE DECLARATIONS -
! local gfx <const> = playdate.graphics
(shorthand, ALLOW TO REUSE GFX INSTEAD FULL PLAYDATE.GRAPHICS)

! local (variable) = nil
(variable to be accessed later on in the code) - SIMILAR TO [PUBLIC INT (VARIABLE)] IN UNITY C#

! LOCAL (VARIABLE) = (amount)
(EX: PLAYER SPEED, SCOREBOARD ~~etc~~, TIMER, ETC)

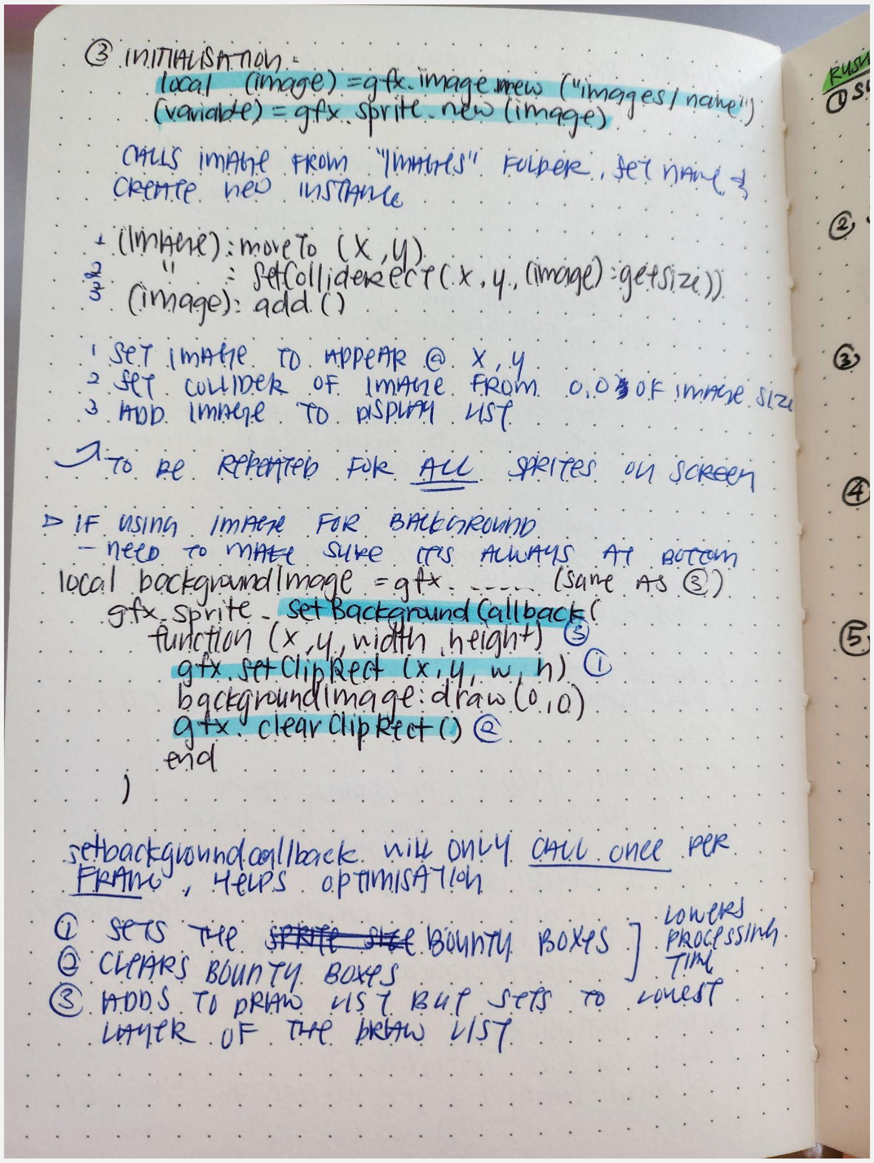
DECLARING SPEED OF PLAYER

DECLARING START SCORE OF SCORE BOARD

* TIMER DURATION

* TIMER OR OTHER MAY NOT NECESSARILY NEED TO BE DECLARED FIRST AS IT MAY CHANGE DEPENDING TO THE DIFFERENT TIMERS (IF MORE)

③ FUNCTION INITIALIZATION
! local function initialize()
- local (image) = gfx.image.new("images/name")



③ INITIALISATION:

```
local (image) = gfx.image.new ("images/name")
(variable) = gfx.sprite.new (image)
```

CALLS IMAGE FROM "IMAGES" FOLDER, SET NAME & CREATE NEW INSTANCE

```
1 (image):moveTo (X, Y)
2   "    : setCollideRect(x, y, (image):getSize())
3 (image): add ()
```

1 SET IMAGE TO APPEAR @ X, Y
2 SET COLLIDER OF IMAGE FROM 0,0 OF IMAGE SIZE
3 ADD IMAGE TO DISPLAY LIST

TO BE REPEATED FOR ALL SPRITES ON SCREEN

▷ IF USING IMAGE FOR BACKGROUND
– NEED TO MAKE SURE IT'S ALWAYS AT BOTTOM

```
local backgroundImage = gfx. ..... (same as ③)
  gfx.sprite.setBackgroundCallback( ③
    function (x, y, width, height)
      gfx.setClipRect (x, y, w, h) ①
      backgroundImage:draw(0,0)
      gfx.clearClipRect () ②
    end
  )
```

setbackgroundcallback WILL ONLY CALL ONCE PER FRAME, HELPS OPTIMISATION

① SETS THE ~~SPRITE SIZE~~ BOUNTY BOXES] LOWERS PROCESSING TIME
② CLEARS BOUNTY BOXES
③ ADDS TO DRAW LIST BUT SETS TO LOWEST LAYER OF THE DRAW LIST

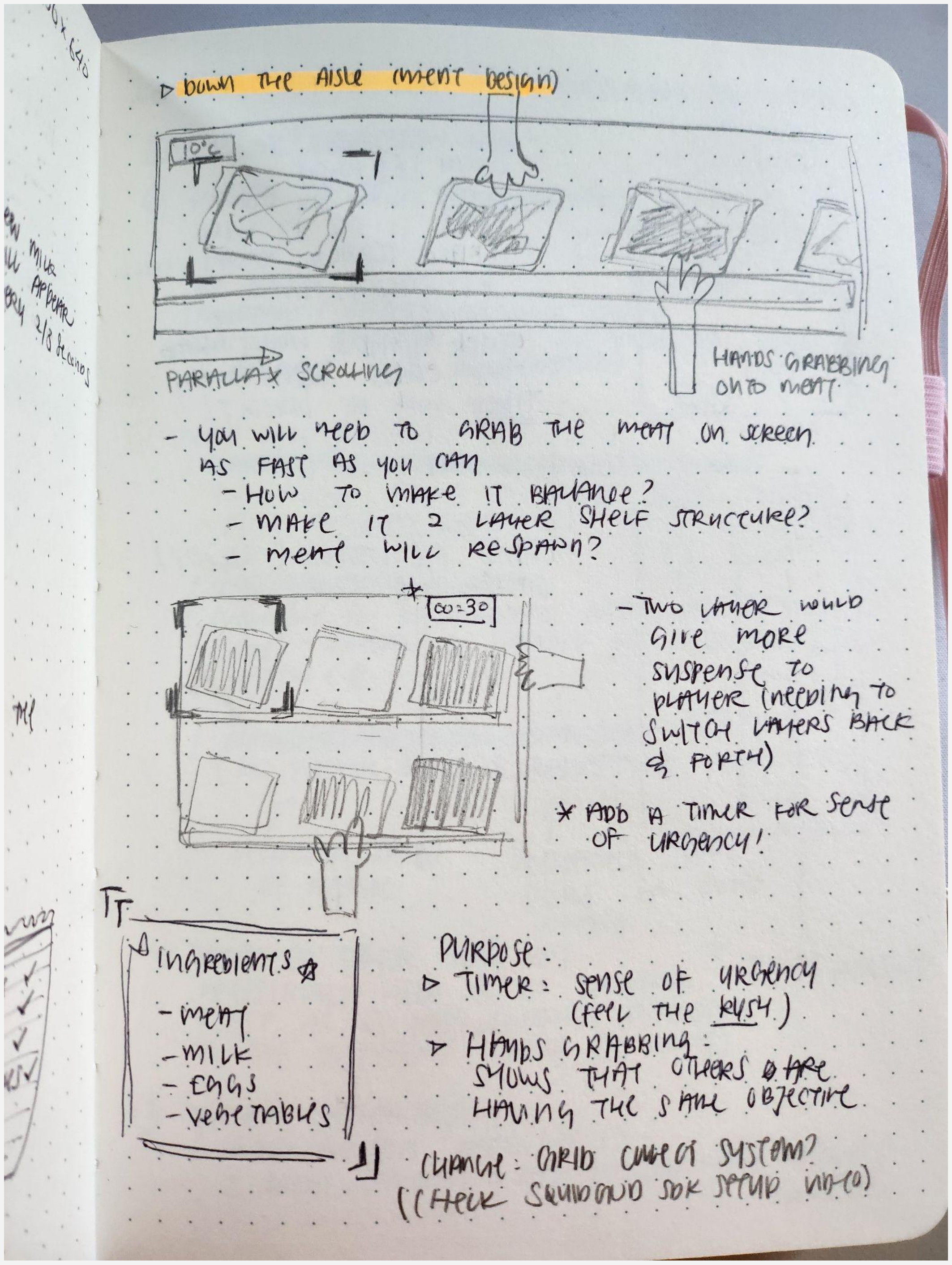
Figure 12-13: Image of notebook of notes from tutorial & structure of coding in Lua

3. Mechanics

3.1. Initial Ideas

- Down the Aisle

Down the Aisle is a parallax scrolling concept for the Playdate console version. My idea was to utilise the Playdate crank to scroll down the aisle while you have to press A to select through the fresh and rotten ingredients. While moving down the aisle you'll encounter elongated hands attempting to grab the ingredients.



▷ Down the Aisle (Meat Design)

- You will need to grab the meat on screen as fast as you can
  - How to make it balance?
  - Make it 2 layer shelf structure?
  - Meat will respawn?

- Two layer would give more suspense to player (needing to switch layers back & forth)

* Add a timer for sense of urgency!

Purpose:
▷ Timer: sense of urgency (feel the rush)
▷ Hands grabbing: shows that others ~~are~~ having the same objective

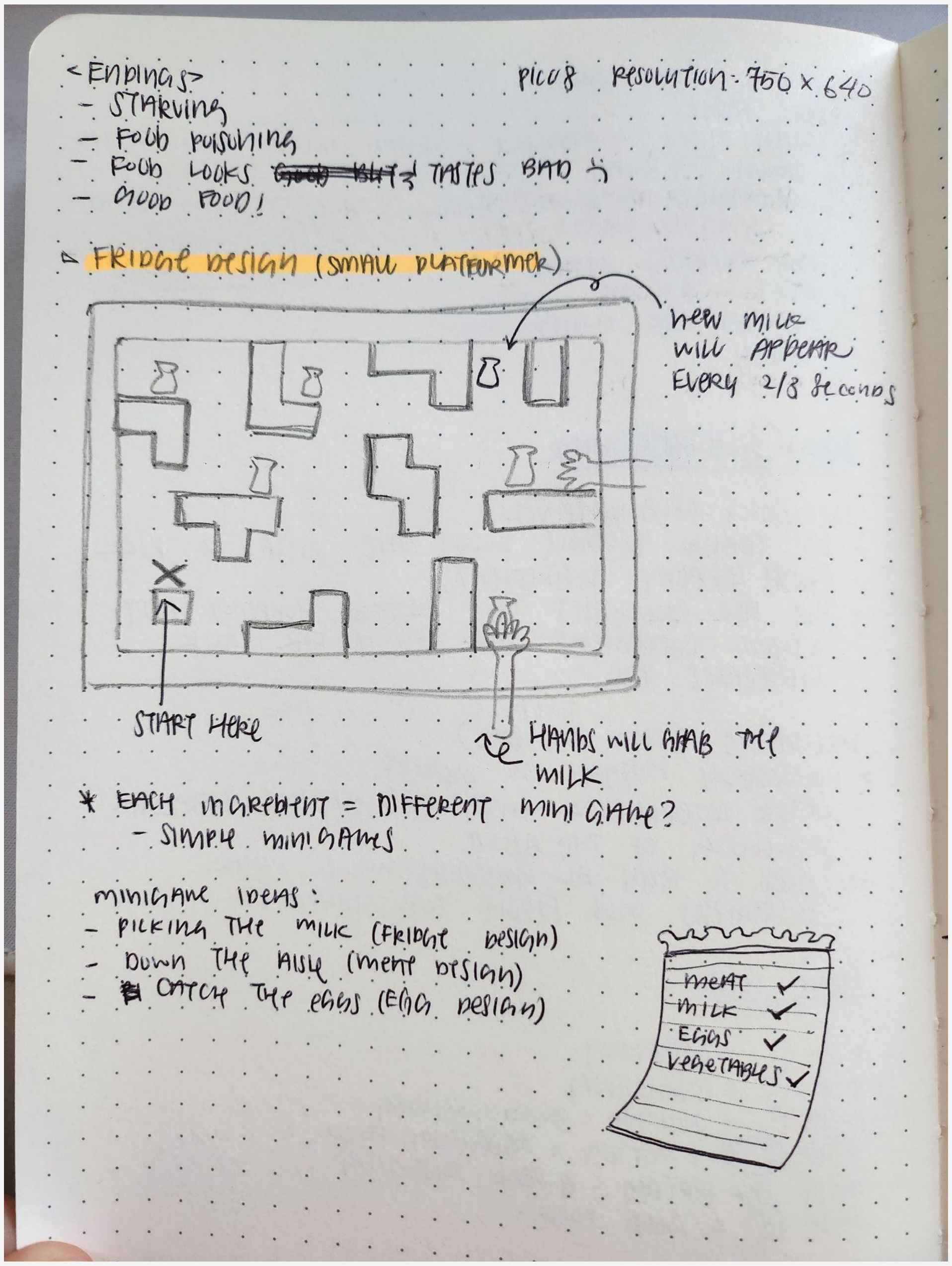
Change: grid collect system?
((check squidguid sdk setup info))

Figure 14: Image of notebook of initial game ideas: Down the Aisle

- Top-down

In the top-down game mode, players are tasked with a frantic race against time and grasping
oO

hands to collect as many ingredients as possible. This mode demands keen attention to multiple elements simultaneously, making it potentially the most challenging among the options, despite its straightforward concept.



<ENDINGS>
- STARVING
- FOOD POISONING
- FOOD LOOKS ~~GOOD BUT~~ & TASTES BAD
- GOOD FOOD!

PICO8 RESOLUTION: 750 x 640

▷ FRIDGE DESIGN (SMALL PLATFORMER)

* EACH INGREDIENT = DIFFERENT MINI GAME?
  - SIMPLE MINIGAMES

MINIGAME IDEAS:
- PICKING THE MILK (FRIDGE DESIGN)
- DOWN THE AISLE (MEAT DESIGN)
- CATCH THE EGGS (EGG DESIGN)

Figure 15: Image of notebook of initial game ideas: Small Platformer

- Whack-A-Mole

Whack-A-Mole presents a unique twist, where players are tasked with pulling ingredients (potatoes) out of the ground instead of simply whacking them. These ingredients appear for a short period before disappearing again. As you search for the ingredients, you'll need to watch out for other customers who may try to snatch them right from under your nose.

- Tug of War (Squid Game Version)

I tend to find myself finding inspirations from movies, TV shows or even narrative from books occasionally. An event that striked an interest in me from a very memorable TV show was the Tug of War scene from Squid Game. The Tug of War scene was very impactful as you knew the severity of their actions could lead to major consequences. I wanted to implement similar techniques (but less deadly) with a twist to fit the game.

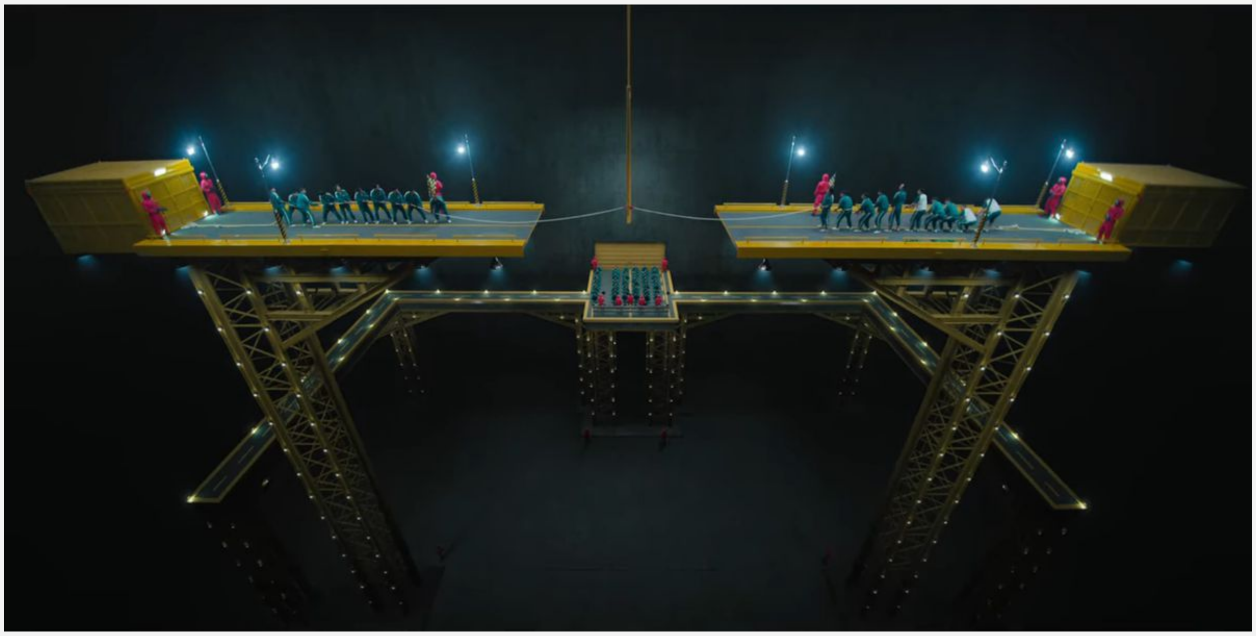


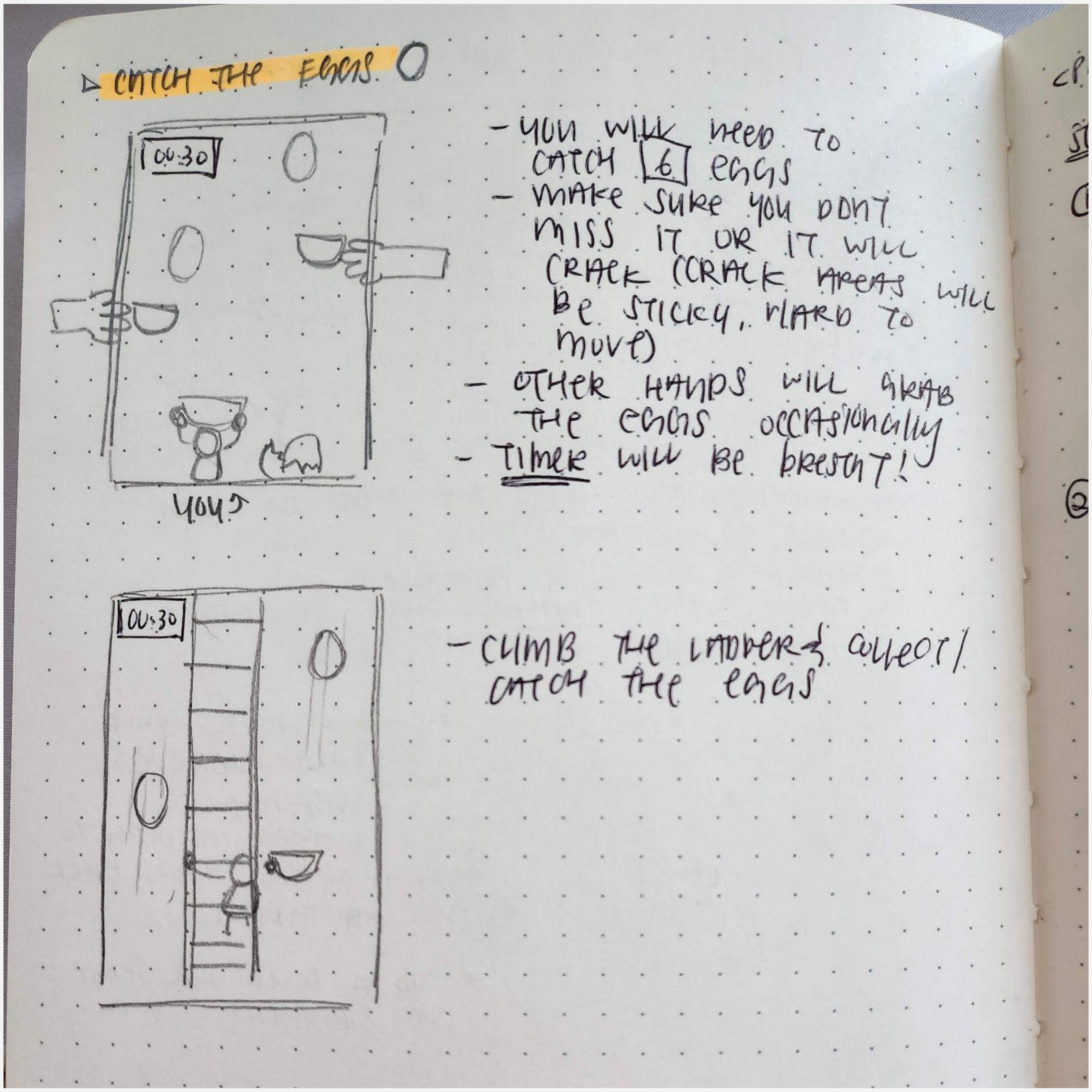
Figure 16: Tug of War scene from Squid Game

In Squid Game, the game worked by pitting two teams against each other in a fierce tug-of-war. The teams had to exert their maximum strength, pulling on a suspended string with all their might. The tension heightened as the string swayed towards one team, and in an instant, a sharp blade sliced through it, plunging the losing team into a pit of certain doom. Gruesome as it may sound, I found the concept incredibly captivating (and no, I'm not a psychopath, just a fan of survival and horror thrillers).

To make it work for Rush Hour, instead of having the player potentially fall into their impending doom, the player is tasked to use the Playdate crank to pull the resistance of the rope against the customers on the other side. There will be a timer for the game where the players

- Catch the Eggs

Catch the Eggs presents a straightforward vertical falling game where the objective is to skillfully catch falling eggs. Players must carefully aim for the designated drop positions of the eggs and patiently await their descent into the bowl they are carrying.



▷ CATCH THE EGGS O

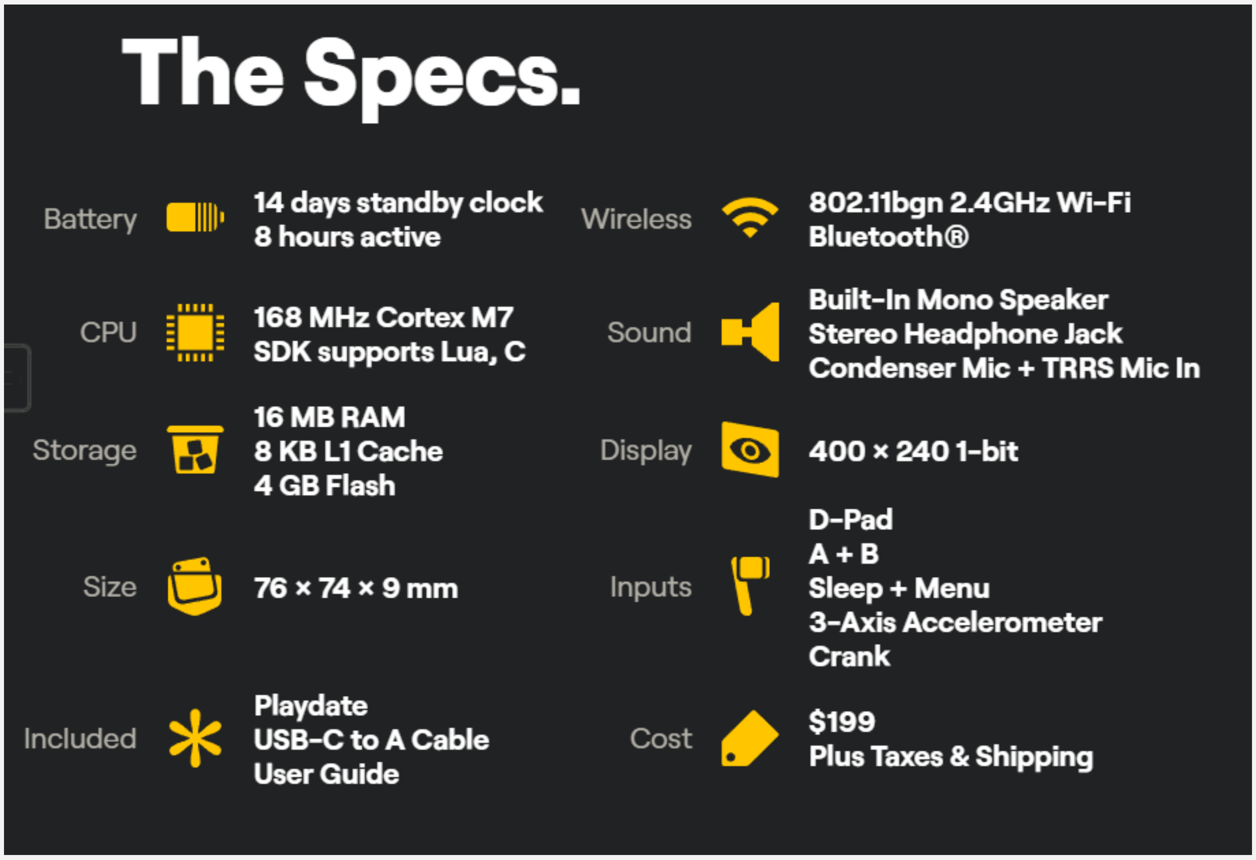
- You will need to catch $\boxed{6}$ eggs
- Make sure you dont miss it or it will crack (crack ~~areas~~ will be sticky, hard to move)
- Other hands will grab the eggs occasionally
- Timer will be present!

- Climb the ladder & collect/catch the eggs

Figure 17: Image of notebook of initial game ideas: Catch the Eggs

3.2. Restrictions & Work-around

Many of the mechanics limitations arose from the Playdate console's hardware constraints. Being a compact device with limited capabilities, developing for the Playdate necessitated a significant focus on optimising the game. It entailed breaking down the game into its simplest form while ensuring it remained enjoyable. Consequently, a considerable amount of time was spent troubleshooting and brainstorming solutions to create the game environment without compromising performance.



The Specs.

Battery	14 days standby clock 8 hours active	Wireless	802.11bgn 2.4GHz Wi-Fi Bluetooth®
CPU	168 MHz Cortex M7 SDK supports Lua, C	Sound	Built-In Mono Speaker Stereo Headphone Jack Condenser Mic + TRRS Mic In
Storage	16 MB RAM 8 KB L1 Cache 4 GB Flash	Display	400 × 240 1-bit
Size	76 × 74 × 9 mm	Inputs	D-Pad A + B Sleep + Menu 3-Axis Accelerometer Crank
Included	Playdate USB-C to A Cable User Guide	Cost	$199 Plus Taxes & Shipping

Figure 18: Hardware specs of Playdate

3.3. Assistance

Given the novelty of the Playdate device, seeking assistance in this specialised field necessitates connecting with individuals who possess expertise in this domain. Consequently, I dedicate a significant portion of my time to immersing myself in the Playdate and SquidGod's Discord servers. By actively observing the ongoing development of other games and engaging with the community, I glean valuable insights and knowledge. The Playdate community is notably supportive and welcoming, readily extending their aid to fellow developers venturing into Playdate game development.

4. Narrative

4.1. Selection of Topic

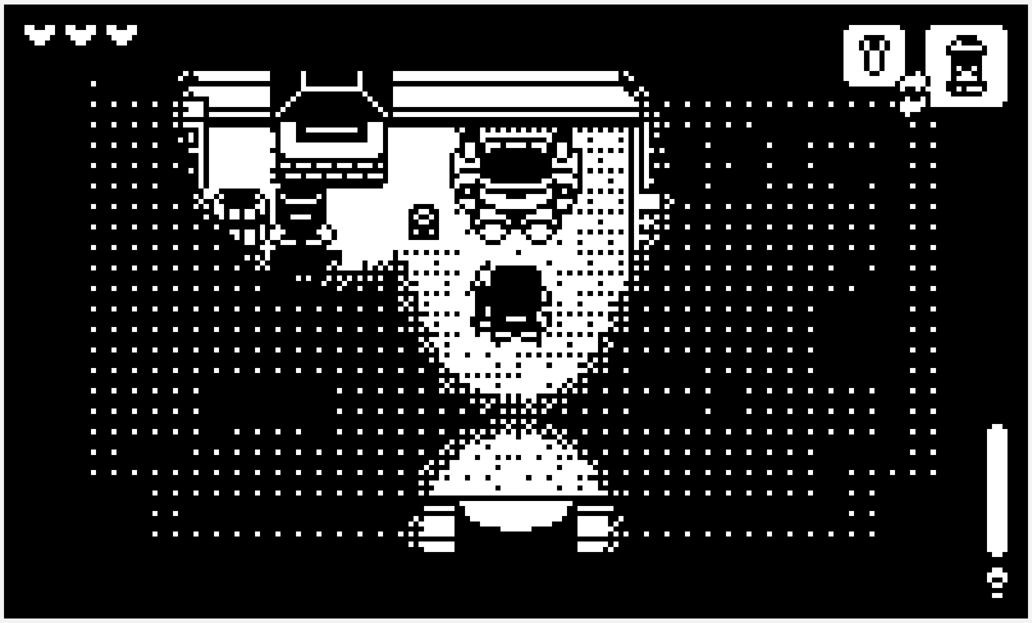
The chosen theme for my project is "Slice of Life" with a touch of autobiographical elements. I aimed to capture a relatable experience that resonates with a wide audience, and grocery shopping emerged as an ideal situation. This mundane activity is a daily necessity for all of us, and at times, we find ourselves compelled to navigate the aisles during peak hours, whether in the morning or evening.

Drawing from my personal experiences, I have often been characterised as having a heightened sense of urgency, which undeniably induces stress. I aspired to capture this sensation within the game, offering players a glimpse into my daily emotional landscape. To achieve this, I incorporated a **timed collection system**, introducing a **timer** and a **branching narrative** that entails the consequences of failed attempts at procuring the required ingredients. This mechanism adds a real sense of urgency, pushing players to tackle the game's obstacles within a limited timeframe.

5. Aesthetics

5.1. Initial Ideas & Selected

From the beginning, I envisioned a simplistic aesthetic for this project. Given the focus that I will be developing on the Playdate, I intended to employ a 1-bit visual style, relying solely on black, white, and transparent sprites. However, as the development progressed, the visual style underwent a subtle transformation to maintain its appealing aesthetics while aligning more seamlessly with the chosen technology.



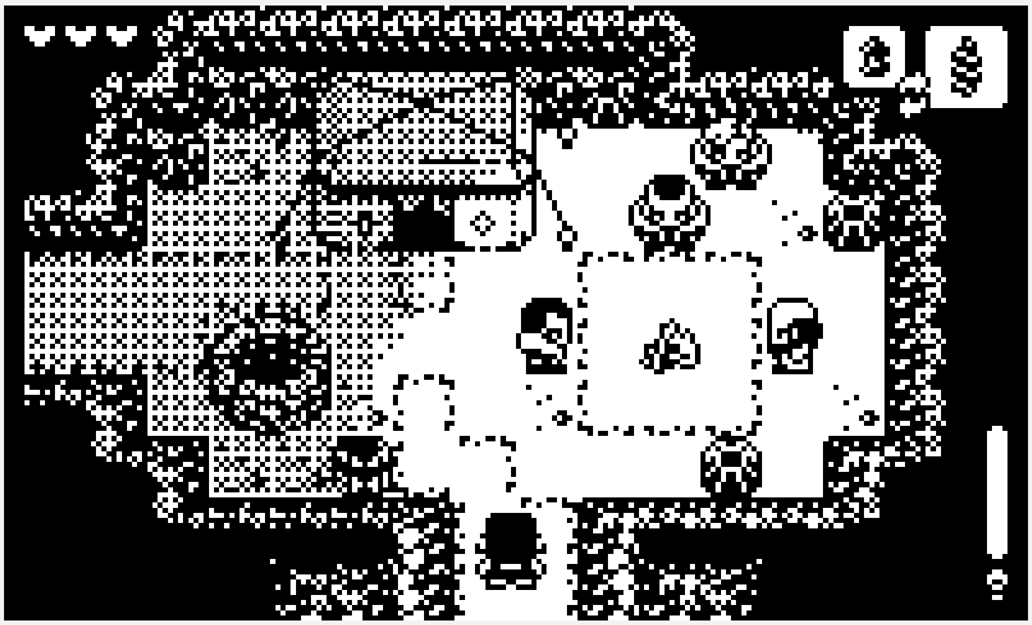


Figure 19-20: Gameplay screenshot of Playdate game: Ratcheteer (Shaun Inman)

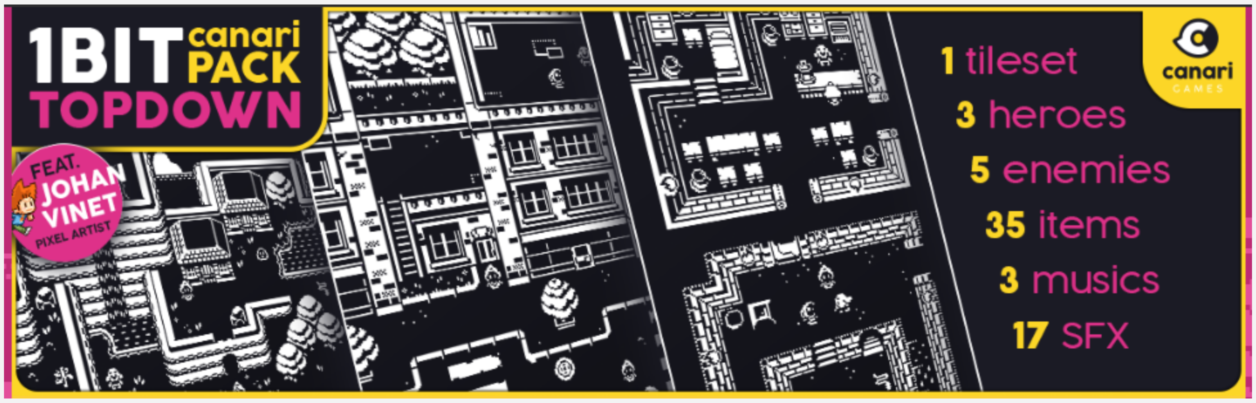
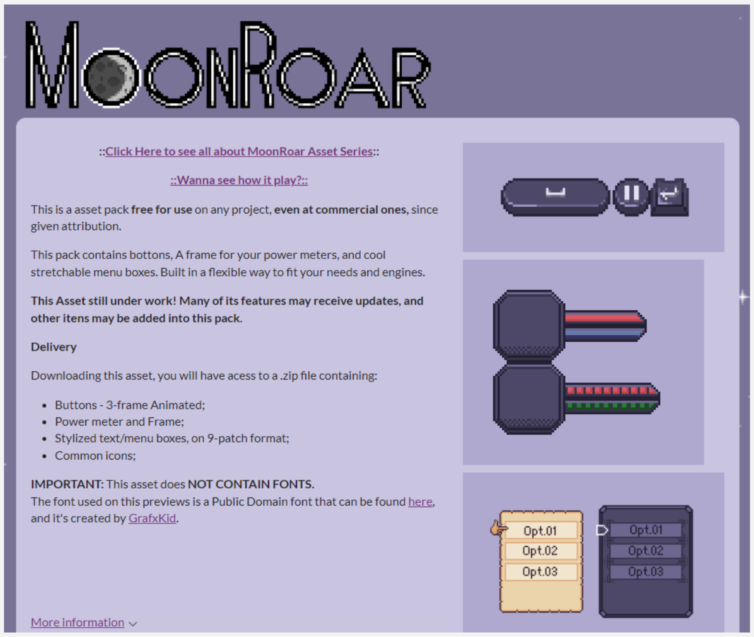


Figure 21: Initial paid asset pack used by CanariGames

Based on the feedback received during the first feedback session, it had become apparent that the aesthetics no longer aligned seamlessly with the intended technology. The compatibility of the aesthetics with the Playdate was primarily attributed to the hardware limitations and platform suitability. Considering my aim to maintain a similar visual style, a decision was made to transition towards a pixel-art style. Moreover, given the limited remaining time, I opted to leverage the expertise of fellow creators on itch.io and incorporate their premium assets.

Modern exteriors
buildings
vehicles
animations
daily updated
Modern exteriors - RPG Tileset [16X16]
$2 -50%
16x16 tilesets for your top-down game
Modern interiors
furniture
characters
animations
400/400
daily updates
Modern interiors - RPG Tileset [16X16]
-50%
16x16 tilesets for your top-down game

Figure 22: Paid assets by LimeZu



::Click Here to see all about MoonRoar Asset Series::

::Wanna see how it play?::

This is a asset pack **free for use** on any project, **even at commercial ones,** since given attribution.

This pack contains bottons, A frame for your power meters, and cool stretchable menu boxes. Built in a flexible way to fit your needs and engines.

This Asset still under work! Many of its features may receive updates, and other itens may be added into this pack.

Delivery

Downloading this asset, you will have acess to a .zip file containing:

- Buttons - 3-frame Animated;
- Power meter and Frame;
- Stylized text/menu boxes, on 9-patch format;
- Common icons;

IMPORTANT: This asset does **NOT CONTAIN FONTS.**
The font used on this previews is a Public Domain font that can be found here, and it's created by GrafxKid.

More information

Figure 23: Free assets by MoonRoar

6. Major Changes in Choice of Technology

6.1. Limitations/Drawbacks & Development Changes

The most significant change I had to make during the development process was the decision to transition the game entirely from one platform to another. Initially, I had ambitious plans for the Playdate version of the game, but unfortunately, I faced numerous obstacles that prevented me from making substantial progress. It became apparent that my expectations were overly ambitious, leading to my downfall. I had underestimated the time available for development and the level of assistance required, which resulted in constant need for guidance.

The decision to switch to Unity was not taken lightly, as I was genuinely disappointed about not being able to create a Playdate game. I believed that the Playdate would have offered a unique and captivating gameplay experience, with its fascinating technology. However, considering my familiarity and the available resources, Unity emerged as the most practical choice. Its vast array of resources and the ability to exchange knowledge among friends proved invaluable.

While this particular endeavour did not result in a Playdate game, I made a point to carefully document the limitations and drawbacks I encountered while working with the Playdate SDK. This will prove beneficial in the future when I inevitably return to working with the platform.

One major drawback I identified is the absence of a "Tilemap Collider 2D" system akin to Unity. Constructing a map with a ceiling becomes intricate, requiring either intricate sprite manipulation and construction within VS Code using pixel positioning or utilising alternative tools such as Level Design Tool Kit (LDTK). It is crucial to thoroughly research and prepare for such challenges in future projects that heavily rely on a robust collision system.

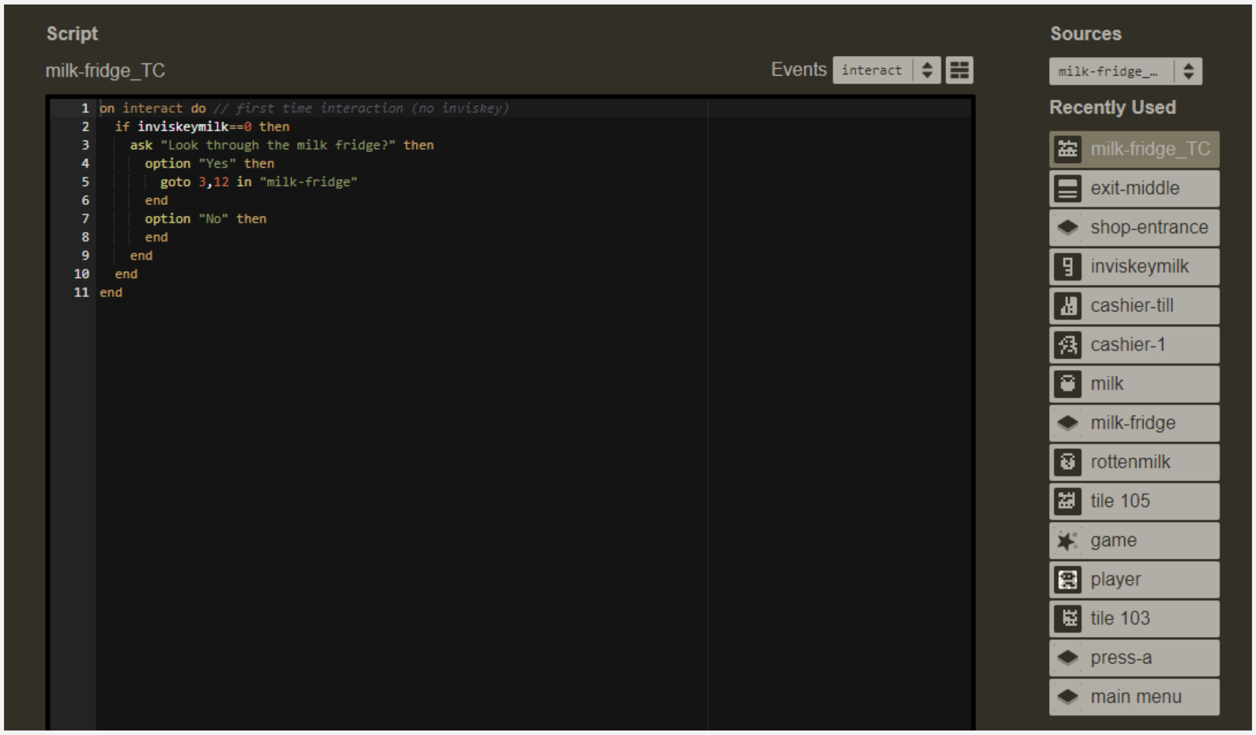
7. Project Process

7.1. Before Starting Week - Game Jam #1 (Pulp Editor Development)

Rush Hour is a game which I worked on during the first week of this term. We were to explore the elements of slice of life and autobiography in video games as a narrative meaning. My plan was to take this as a practice round for the upcoming game for the final project. Initially, I wanted to start off to create a PICO-8 game, since thinking that PICO-8 uses a similar programming language as for the Playdate on the SDK. After consulting with Richard regarding the decision, Richard advised that it would be easier to directly develop for the Playdate instead of trying it out on the PICO-8 and switching afterwards, as PICO-8's Lua language is not exactly the same as what Playdate Lua is like.

Since I only had a week to work on this game jam, I decided to look around alternatives on how I can develop a simple prototype for the Playdate in a week's time. It is at this time I came across the Playdate Pulp Editor. Pulp Editor is a Bitsy-Inspired browser based game maker, it allows game developers of any skill level to utilise its simple yet feature packed system to create a Playdate game. After looking through some examples of Playdate games created on the Pulp Editor, it is shown that Pulp Editor can be used to make a fully fledged game with a similar complexity of what the SDK can do. It is at this time I decided to give the Pulp Editor a try and I learn it's "simple" looking PulpScript.

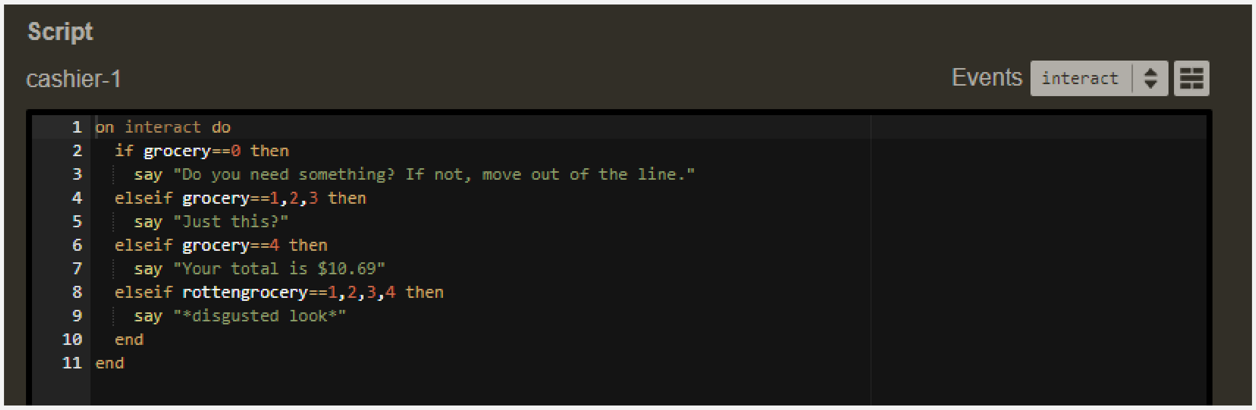
The whole concept of PulpScript is a very dumb down version of if else statement used in C#, you start with *on interact*, when the player interacts with an item, the game then checks a condition, for example *if inviskeymilk=0 then*, if player does not hold any inviskeymilk, an action you set may proceed in game. In this case, I set the player to go to a specific location in a different room.



```
on interact do // first time interaction (no inviskey)
  if inviskeymilk==0 then
    ask "Look through the milk fridge?" then
      option "Yes" then
        goto 3,12 in "milk-fridge"
      end
      option "No" then
      end
    end
  end
end
```

Figure 24: PulpScript game scene manager system

Armed with this knowledge, I was able to incorporate it into an inventory system that would be triggered when the player interacts with the NPC, specifically the cashier. By leveraging this system, the cashier could effectively assess the quantity of fresh and rotten groceries in the player's inventory, leading to diverse dialogue outcomes based on the specific circumstances.



```
on interact do
  if grocery==0 then
    say "Do you need something? If not, move out of the line."
  elseif grocery==1,2,3 then
    say "Just this?"
  elseif grocery==4 then
    say "Your total is $10.69"
  elseif rottengrocery==1,2,3,4 then
    say "*disgusted look*"
  end
end
```

Figure 25: PulpScript inventory system

During the game development process, I took the opportunity to delve into the realm of music creation by trying my hand at the song creator feature. I crafted a few uncomplicated yet catchy chimes that could be seamlessly looped within the game. This would then be triggered depending on the room entering or prompted when a specific action has been made.

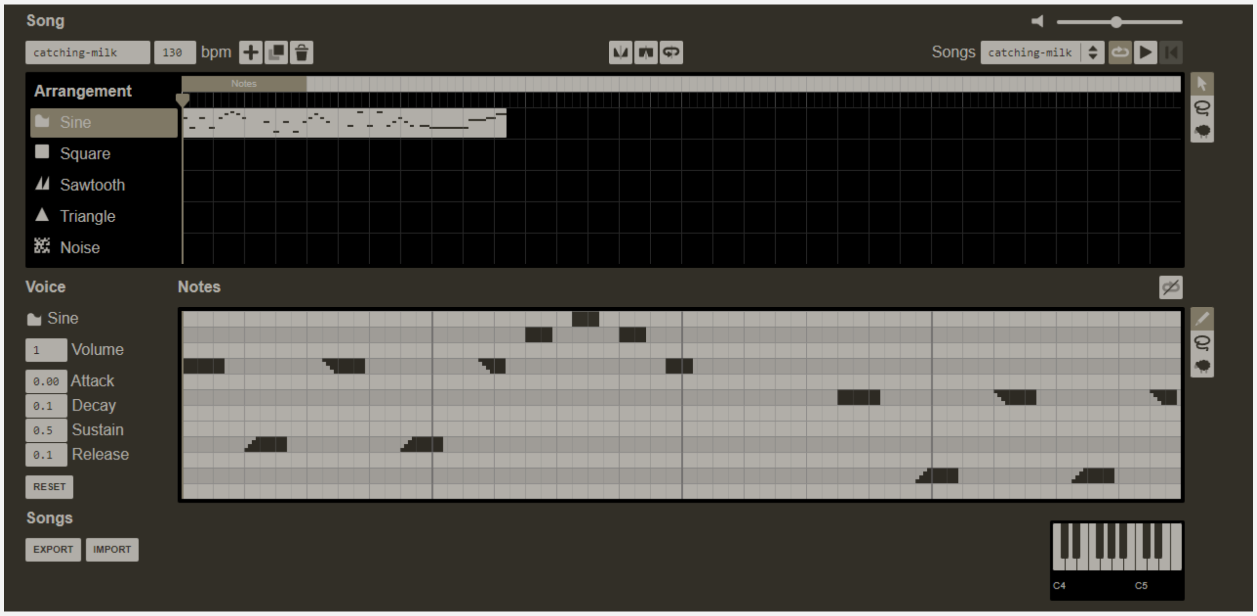


Figure 26: PulpScript music creator

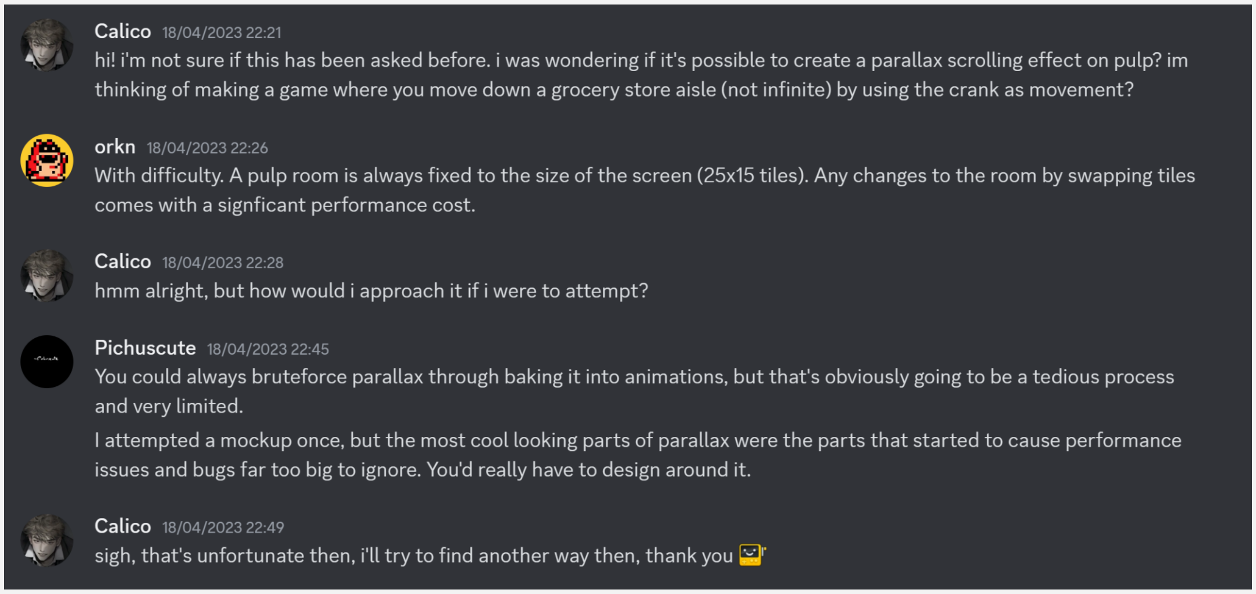
Regrettably, my familiarity with the entirely new system was insufficient to showcase any core mechanics by the end of the game jam. Nonetheless, my commitment to the project extended beyond the jam, motivating me to persistently learn and unravel the intricacies of the Pulp Editor even after the event concluded. Recognizing my desire to continue working on this game, I invested additional effort into comprehending its inner workings.

Brief Reflection

Despite the limited timeframe of the game jam, numerous captivating ideas and concepts emerged that motivated me to carry them forward into the main project. As a result, I decided to bypass the concepting stage for the time being, allowing me to allocate more time to learn the necessary programming languages, systems, and overall development techniques required to bring the game to life using the Playdate SDK. This strategic adjustment enables me to embark on the subsequent development phase with a solid foundation and a clearer vision for the game.

7.2. Week 1-4 (Playdate SDK Development)

Throughout these two weeks, my primary objective was to determine the feasibility of implementing the mechanics I envisioned using the Pulp Editor. To gain insights, I sought the help of the Discord community and specifically inquired about the viability of incorporating a parallax scrolling effect. Unfortunately, the responses I received from the community indicated that achieving this effect within the confines of the Pulp Editor seemed implausible.



Calico 18/04/2023 22:21
hi! i'm not sure if this has been asked before. i was wondering if it's possible to create a parallax scrolling effect on pulp? im thinking of making a game where you move down a grocery store aisle (not infinite) by using the crank as movement?

orkn 18/04/2023 22:26
With difficulty. A pulp room is always fixed to the size of the screen (25x15 tiles). Any changes to the room by swapping tiles comes with a signficant performance cost.

Calico 18/04/2023 22:28
hmm alright, but how would i approach it if i were to attempt?

Pichuscute 18/04/2023 22:45
You could always bruteforce parallax through baking it into animations, but that's obviously going to be a tedious process and very limited.
I attempted a mockup once, but the most cool looking parts of parallax were the parts that started to cause performance issues and bugs far too big to ignore. You'd really have to design around it.

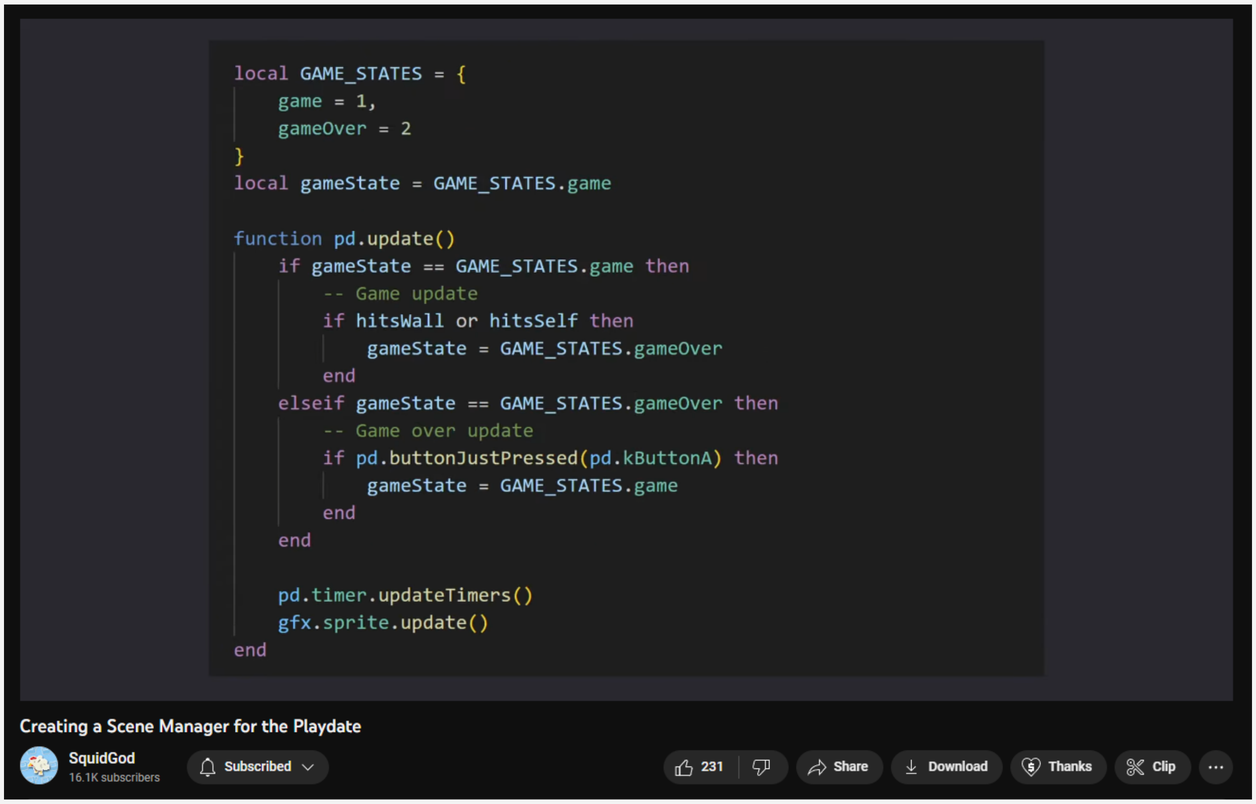
Calico 18/04/2023 22:49
sigh, that's unfortunate then, i'll try to find another way then, thank you

Figure 27: Screenshot of assistance request on Playdate Discord Server

Realising that the grand scope of my plans exceeded the capabilities of the Pulp Editor, I made the pivotal decision to shift my focus towards utilising the SDK. However, this transition brought forth a new challenge as it entailed working with Lua, a programming language with which I still felt relatively unfamiliar despite watching numerous tutorials.

To overcome this obstacle, I turned to SquidGod's Github repository, where I meticulously explored his previous project codes. By thoroughly examining and deconstructing his work, I gained valuable insights into Lua programming and its application within the Playdate ecosystem. This process proved instrumental in enhancing my understanding of Lua and its implementation in game development.

Given the frequent scene changes required in the game, my initial focus revolved around developing a scene manager system. Following SquidGod's tutorial, I acquired the necessary knowledge to create a basic wipe transition. Although I ultimately decided against using the wipe transition, as it didn't convey the desired effect, the learning experience itself proved intriguing.

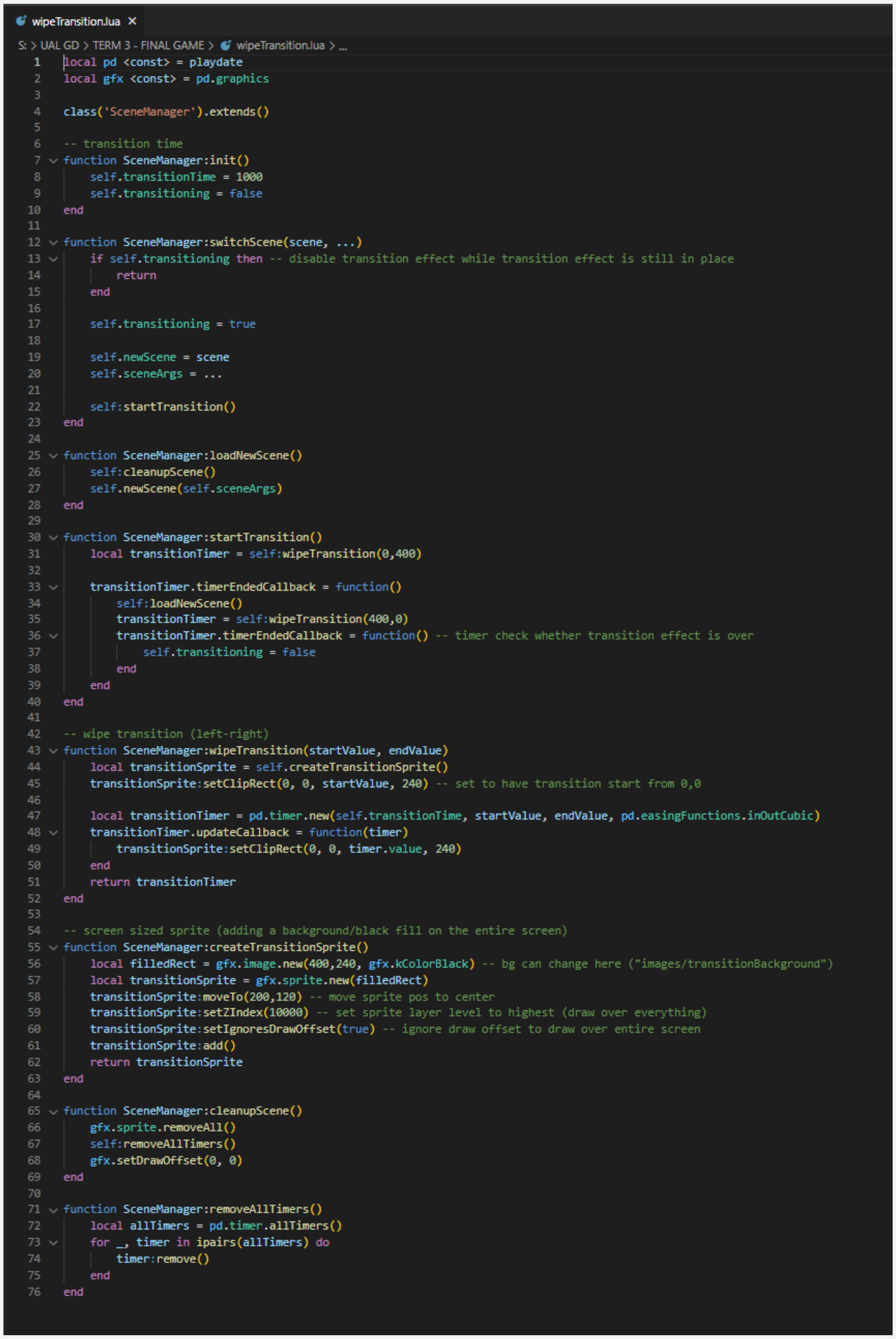


```lua
local GAME_STATES = {
    game = 1,
    gameOver = 2
}
local gameState = GAME_STATES.game

function pd.update()
    if gameState == GAME_STATES.game then
        -- Game update
        if hitsWall or hitsSelf then
            gameState = GAME_STATES.gameOver
        end
    elseif gameState == GAME_STATES.gameOver then
        -- Game over update
        if pd.buttonJustPressed(pd.kButtonA) then
            gameState = GAME_STATES.game
        end
    end

    pd.timer.updateTimers()
    gfx.sprite.update()
end
```

Creating a Scene Manager for the Playdate

Figure 28: SquidGod's video on Scene Manager and transitions

Continuing my search for suitable scene transitions, I stumbled upon a pinhole transition that perfectly aligned with my requirements. It provided the desired visual effect, prompting me to incorporate it into the game seamlessly.

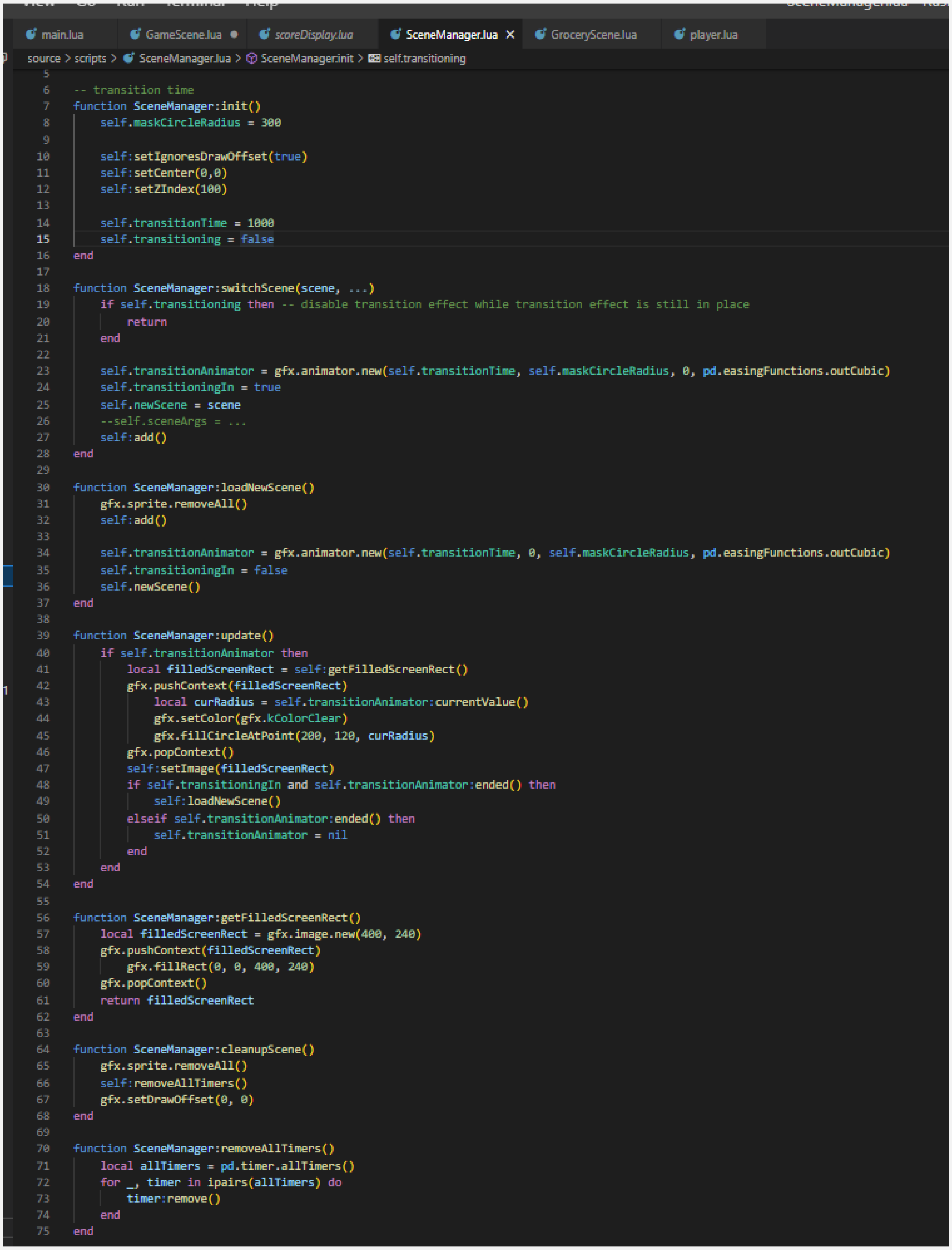


wipeTransition.lua ×

S: > UAL GD > TERM 3 - FINAL GAME > wipeTransition.lua > ...

```lua
local pd <const> = playdate
local gfx <const> = pd.graphics

class('SceneManager').extends()

-- transition time
function SceneManager:init()
    self.transitionTime = 1000
    self.transitioning = false
end

function SceneManager:switchScene(scene, ...)
    if self.transitioning then -- disable transition effect while transition effect is still in place
        return
    end

    self.transitioning = true

    self.newScene = scene
    self.sceneArgs = ...

    self:startTransition()
end

function SceneManager:loadNewScene()
    self:cleanupScene()
    self.newScene(self.sceneArgs)
end

function SceneManager:startTransition()
    local transitionTimer = self:wipeTransition(0,400)

    transitionTimer.timerEndedCallback = function()
        self:loadNewScene()
        transitionTimer = self:wipeTransition(400,0)
        transitionTimer.timerEndedCallback = function() -- timer check whether transition effect is over
            self.transitioning = false
        end
    end
end

-- wipe transition (left-right)
function SceneManager:wipeTransition(startValue, endValue)
    local transitionSprite = self.createTransitionSprite()
    transitionSprite:setClipRect(0, 0, startValue, 240) -- set to have transition start from 0,0

    local transitionTimer = pd.timer.new(self.transitionTime, startValue, endValue, pd.easingFunctions.inOutCubic)
    transitionTimer.updateCallback = function(timer)
        transitionSprite:setClipRect(0, 0, timer.value, 240)
    end
    return transitionTimer
end

-- screen sized sprite (adding a background/black fill on the entire screen)
function SceneManager:createTransitionSprite()
    local filledRect = gfx.image.new(400,240, gfx.kColorBlack) -- bg can change here ("images/transitionBackground")
    local transitionSprite = gfx.sprite.new(filledRect)
    transitionSprite:moveTo(200,120) -- move sprite pos to center
    transitionSprite:setZIndex(10000) -- set sprite layer level to highest (draw over everything)
    transitionSprite:setIgnoresDrawOffset(true) -- ignore draw offset to draw over entire screen
    transitionSprite:add()
    return transitionSprite
end

function SceneManager:cleanupScene()
    gfx.sprite.removeAll()
    self:removeAllTimers()
    gfx.setDrawOffset(0, 0)
end

function SceneManager:removeAllTimers()
    local allTimers = pd.timer.allTimers()
    for _, timer in ipairs(allTimers) do
        timer:remove()
    end
end
```

Figure 29: Wipe transition script



main.lua GameScene.lua scoreDisplay.lua SceneManager.lua GroceryScene.lua player.lua

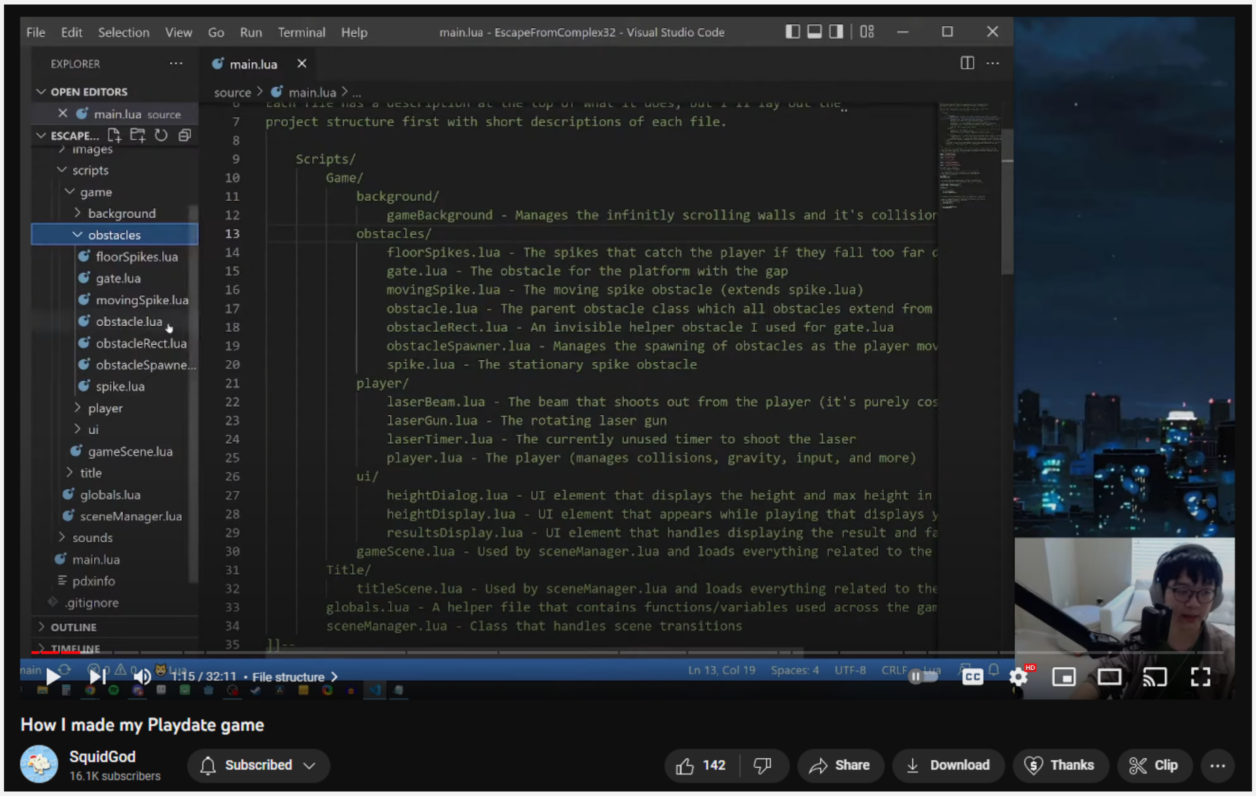
source > scripts > SceneManager.lua > SceneManager:init > self.transitioning

```lua

-- transition time
function SceneManager:init()
    self.maskCircleRadius = 300

    self:setIgnoresDrawOffset(true)
    self:setCenter(0,0)
    self:setZIndex(100)

    self.transitionTime = 1000
    self.transitioning = false
end

function SceneManager:switchScene(scene, ...)
    if self.transitioning then -- disable transition effect while transition effect is still in place
        return
    end

    self.transitionAnimator = gfx.animator.new(self.transitionTime, self.maskCircleRadius, 0, pd.easingFunctions.outCubic)
    self.transitioningIn = true
    self.newScene = scene
    --self.sceneArgs = ...
    self:add()
end

function SceneManager:loadNewScene()
    gfx.sprite.removeAll()
    self:add()

    self.transitionAnimator = gfx.animator.new(self.transitionTime, 0, self.maskCircleRadius, pd.easingFunctions.outCubic)
    self.transitioningIn = false
    self.newScene()
end

function SceneManager:update()
    if self.transitionAnimator then
        local filledScreenRect = self:getFilledScreenRect()
        gfx.pushContext(filledScreenRect)
            local curRadius = self.transitionAnimator:currentValue()
            gfx.setColor(gfx.kColorClear)
            gfx.fillCircleAtPoint(200, 120, curRadius)
        gfx.popContext()
        self:setImage(filledScreenRect)
        if self.transitioningIn and self.transitionAnimator:ended() then
            self:loadNewScene()
        elseif self.transitionAnimator:ended() then
            self.transitionAnimator = nil
        end
    end
end

function SceneManager:getFilledScreenRect()
    local filledScreenRect = gfx.image.new(400, 240)
    gfx.pushContext(filledScreenRect)
        gfx.fillRect(0, 0, 400, 240)
    gfx.popContext()
    return filledScreenRect
end

function SceneManager:cleanupScene()
    gfx.sprite.removeAll()
    self:removeAllTimers()
    gfx.setDrawOffset(0, 0)
end

function SceneManager:removeAllTimers()
    local allTimers = pd.timer.allTimers()
    for _, timer in ipairs(allTimers) do
        timer:remove()
    end
end
```

Figure 30: Pinhole transition script

The pinhole transition functions by utilising a black rectangle sprite that covers the entire screen. Concurrently, a transparent circular mask dynamically intersects the black sprite in each frame. This clever interplay creates the illusion of a pinhole opening, lending a distinctive visual effect to the transition.

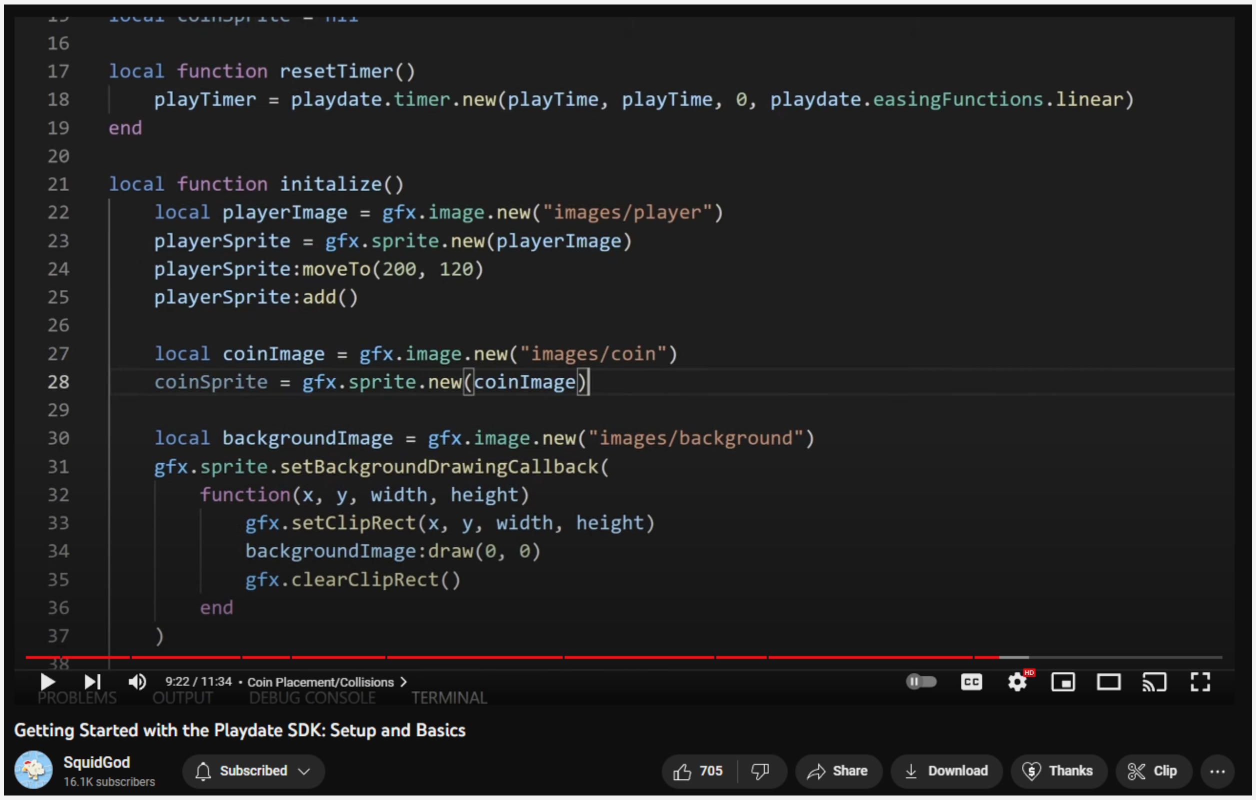
One particular video by SquidGod captivated my attention, and I devoted a significant amount of time to studying its content. In this video, SquidGod provided an insightful explanation of his workflow and structure when developing a Playdate game. A key technique he employed is known as Object-Oriented Programming (OOP). This approach enabled him to compartmentalise various elements such as characters, environments, enemy NPCs, and the score system into distinct scripts. These scripts were then interconnected through a game manager, resulting in a cohesive and organised script system within his project.



```
project structure first with short descriptions of each file.

    Scripts/
        Game/
            background/
                gameBackground - Manages the infinitly scrolling walls and it's collision
            obstacles/
                floorSpikes.lua - The spikes that catch the player if they fall too far d
                gate.lua - The obstacle for the platform with the gap
                movingSpike.lua - The moving spike obstacle (extends spike.lua)
                obstacle.lua - The parent obstacle class which all obstacles extend from
                obstacleRect.lua - An invisible helper obstacle I used for gate.lua
                obstacleSpawner.lua - Manages the spawning of obstacles as the player mov
                spike.lua - The stationary spike obstacle
            player/
                laserBeam.lua - The beam that shoots out from the player (it's purely cos
                laserGun.lua - The rotating laser gun
                laserTimer.lua - The currently unused timer to shoot the laser
                player.lua - The player (manages collisions, gravity, input, and more)
            ui/
                heightDialog.lua - UI element that displays the height and max height in
                heightDisplay.lua - UI element that appears while playing that displays y
                resultsDisplay.lua - UI element that handles displaying the result and fa
            gameScene.lua - Used by sceneManager.lua and loads everything related to the
        Title/
            titleScene.lua - Used by sceneManager.lua and loads everything related to the
        globals.lua - A helper file that contains functions/variables used across the gam
        sceneManager.lua - Class that handles scene transitions
]]--
```

Figure 31: SquidGod's project structure video

Once I had established the foundational elements of the project, I proceeded to develop a straightforward collection system, incorporating a functional timer and score mechanism inspired by SquidGod's tutorial. The tutorial guided me through the creation of a swift timer system that halts player movement once the timer reaches zero. Additionally, I learned to implement a spawner system that randomly generates collectible items within a designated area. Alongside this, I integrated a score system to track the number of coins collected throughout the gameplay experience. Having successfully implemented a spawner, score and timer system that met my expectations and a functioning scene transition system, I now had a solid foundation to build upon. These essential components provided a reliable starting point for further development and refinement of the game.

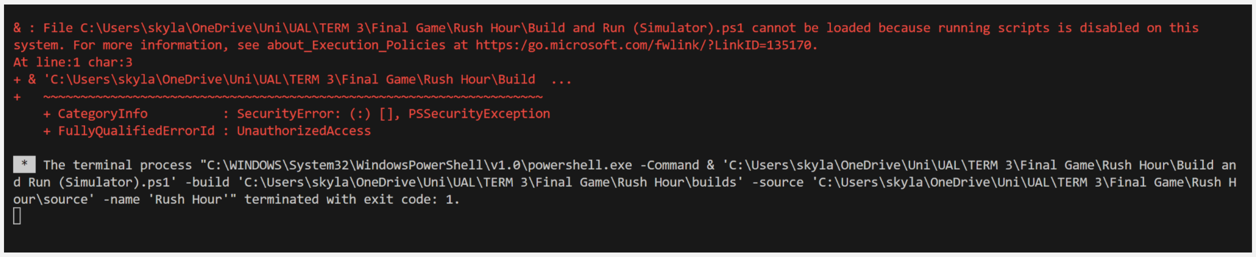


```lua

local function resetTimer()
    playTimer = playdate.timer.new(playTime, playTime, 0, playdate.easingFunctions.linear)
end

local function initalize()
    local playerImage = gfx.image.new("images/player")
    playerSprite = gfx.sprite.new(playerImage)
    playerSprite:moveTo(200, 120)
    playerSprite:add()

    local coinImage = gfx.image.new("images/coin")
    coinSprite = gfx.sprite.new(coinImage)

    local backgroundImage = gfx.image.new("images/background")
    gfx.sprite.setBackgroundDrawingCallback(
        function(x, y, width, height)
            gfx.setClipRect(x, y, width, height)
            backgroundImage:draw(0, 0)
            gfx.clearClipRect()
        end
    )
```

Figure 32: SquidGod's tutorial on development on Playdate

Brief Reflection

Reflecting upon these past few weeks, one aspect I deeply regret is the substantial amount of time I dedicated to resolving the persistent simulator issue I encountered. The Playdate SDK relies on the connection between VS Code and the SDK file itself, but the directory seemed to malfunction, rendering the SDK library inaccessible to any of the scripts. Consequently, this unexpected setback consumed a significant portion of my time, which I had not accounted for in my initial plans.



```
& : File C:\Users\skyla\OneDrive\Uni\UAL\TERM 3\Final Game\Rush Hour\Build and Run (Simulator).ps1 cannot be loaded because running scripts is disabled on this
system. For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:3
+ & 'C:\Users\skyla\OneDrive\Uni\UAL\TERM 3\Final Game\Rush Hour\Build  ...
+   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess

 *  The terminal process "C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe -Command & 'C:\Users\skyla\OneDrive\Uni\UAL\TERM 3\Final Game\Rush Hour\Build an
d Run (Simulator).ps1' -build 'C:\Users\skyla\OneDrive\Uni\UAL\TERM 3\Final Game\Rush Hour\builds' -source 'C:\Users\skyla\OneDrive\Uni\UAL\TERM 3\Final Game\Rush H
our\source' -name 'Rush Hour'" terminated with exit code: 1.
```

Figure 33: Directory failed to be found

7.3. Week 5-6 (Shifting to Unity Development)

As I proceed into Week 5, it is when the stress starts to hit. I felt that the game I was working on was no longer feasible due to the amount of knowledge I would need to learn to develop the game before the time was up. The most significant challenge I encountered during this phase of development was creating an appropriate environment for the functional system I had established in the previous week. I soon realised that collisions in the Playdate SDK function differently compared to Unity, a crucial aspect I had overlooked during my initial planning. This realisation posed a considerable obstacle that required me to reassess and adapt my approach to collision handling. It was only at this time that I realised that creating collision boxes for a whole boxed shape map would require me to use a separate level designing software called: Level Design Tool Kit (LDTK).

The LDTK would aid developers in creating levels within the Playdate much faster than doing it in the manual method. However, using the LDTK would require me to use a whole different library of resources which I do not have prior knowledge to. I was already struggling with prior issues with the game's core mechanics, adding a whole different software and its libraries would require me to hold off all of the existing tasks and focus on it instead. With 2 weeks left before submission. I had to come up with a solution *fast*.

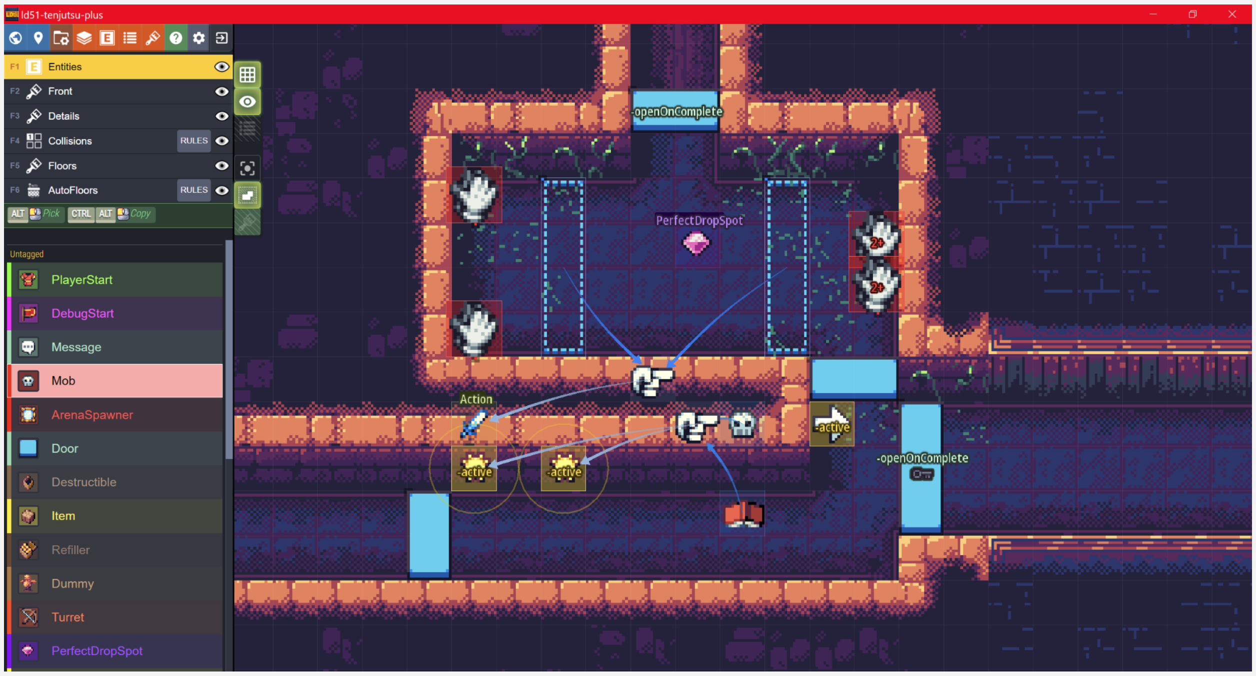
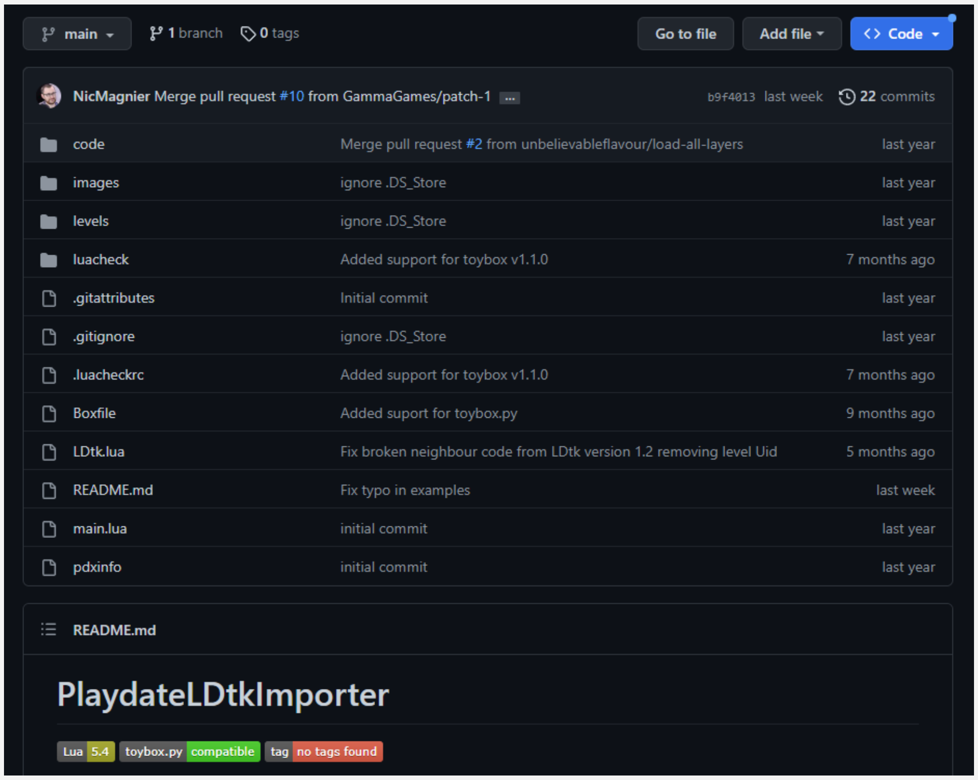


Figure 34: Sample of LDTK workflow



main · 1 branch · 0 tags · Go to file · Add file · Code

NicMagnier Merge pull request #10 from GammaGames/patch-1 … b9f4013 last week 22 commits

code	Merge pull request #2 from unbelievableflavour/load-all-layers	last year
images	ignore .DS_Store	last year
levels	ignore .DS_Store	last year
luacheck	Added support for toybox v1.1.0	7 months ago
.gitattributes	Initial commit	last year
.gitignore	ignore .DS_Store	last year
.luacheckrc	Added support for toybox v1.1.0	7 months ago
Boxfile	Added suport for toybox.py	9 months ago
LDtk.lua	Fix broken neighbour code from LDtk version 1.2 removing level Uid	5 months ago
README.md	Fix typo in examples	last week
main.lua	initial commit	last year
pdxinfo	initial commit	last year

README.md

PlaydateLDtkImporter

Lua 5.4 · toybox.py compatible · tag no tags found

Figure 35: LDTK Importer for the Playdate

This is when I began consulting my friends, Kathi, Anu and Vlad. I knew I had a huge issue with the time frame being so close and I knew I was not able to submit anything at this rate. It is then they suggested that I should retain the idea of the project, but shift the project to Unity instead. I admitted that Unity would definitely solve my issues at a much faster rate as there are plenty more resources and help for developing on Unity. It is then on the same day, I had decided to make the drastic change from developing on the Playdate SDK, to Unity.

I had to replicate the same mechanics quickly, with the placeholder sprites that I had used for the Playdate. After dedicating a weekend to trial and error, I successfully established the foundational core mechanics of the game. The food spawner effectively generated and removed food items after a predetermined duration. Moreover, I implemented a power-up ability that allowed the player to dash towards the food, enabling them to collect it before it despawned. Encouraged by this progress, I presented the prototype to my classmates and engaged in a valuable feedback session. The feedback I received proved to be incredibly insightful, compelling me to make significant aesthetic changes to enhance the overall game experience.

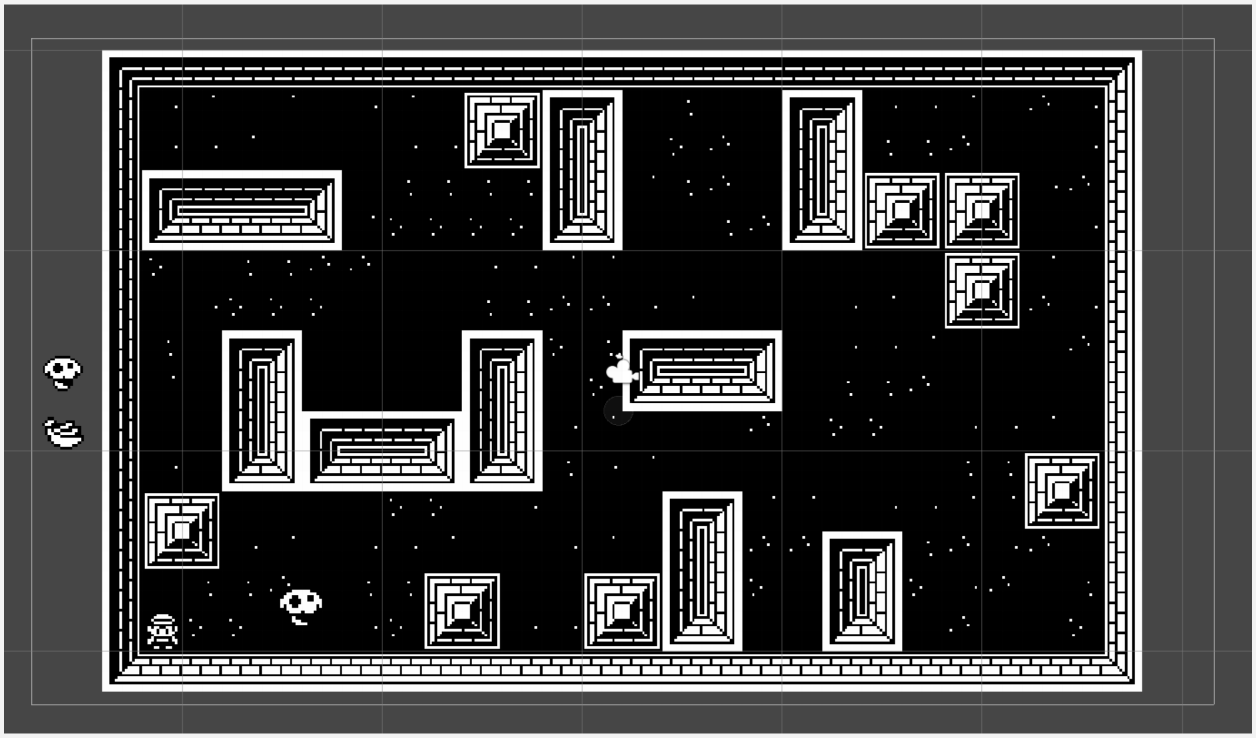


Figure 36: First prototype of Unity version using Playdate aesthetics as placeholder

7.4. Week 7-8

Taking the feedback to heart, I made the bold decision to completely overhaul the game scene, aiming for a fresh visual style. In search of suitable assets, I explored itch.io, where I stumbled upon a captivating grocery shop scene created by LimeZu. This scene exuded the exact ambiance I envisioned, prompting me to purchase the asset pack for a reasonable price of $1.50. Excited by the possibilities, I eagerly started working with the newly acquired assets, eager to bring the envisioned aesthetic to life in the game.

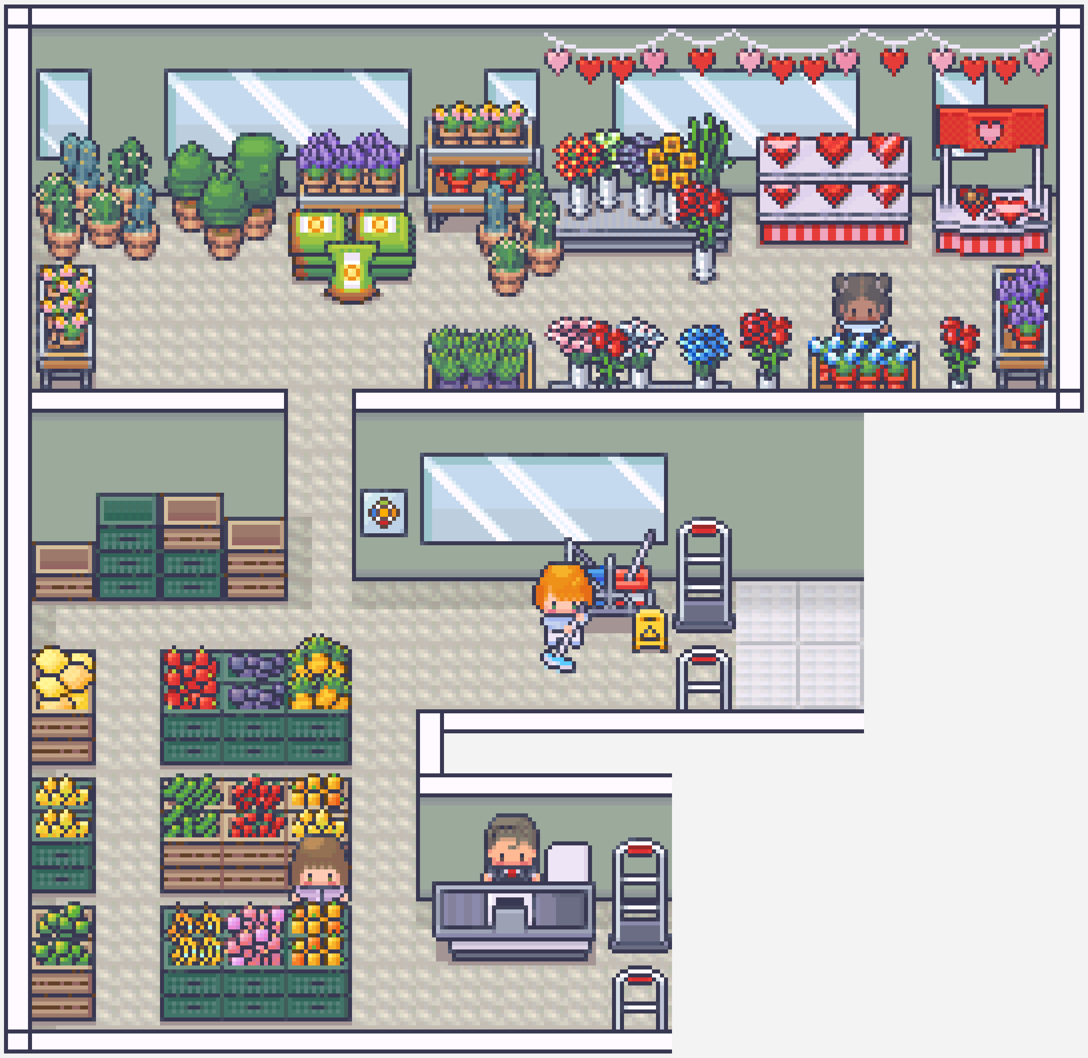


Figure 37: LimeZu's asset pack: Grocery Shop

With the revamped aesthetics and the reintroduction of the food spawner into the scene, the overall ambiance of the game underwent a significant transformation for the better. It now exuded a charming slice-of-life vibe that greatly excited me. However, as I proceeded to place the sprites within the scene, I encountered the familiar challenge of dealing with collisions.

Specifically, I needed to ensure that the food spawned in locations where collisions were present, making them accessible and collectible for the player.

To address this, my initial approach involved creating a script that randomly spawned the items within a designated surface area, while checking whether that particular area had a collision layer mask. This way, I aimed to restrict the spawning to areas where collisions were present, such as the aisles, fridges, and cashier points. Unfortunately, this approach proved ineffective, as the spawner would sometimes generate items outside the defined bounds, rendering the script useless. No matter how I adjusted the script, it failed to adhere to the intended world transform position for spawning within the designated boundaries.

In my quest for a solution, I turned to my friends Kathi and Vlad for their input. Kathi proposed a clever alternative approach: rather than relying on the world transform space, we could utilise pre-defined spawn points and randomise the item placement based on those points. The idea was to create a substantial number of spawn points and have the script select one randomly during each spawn event. This ingenious solution proved to be successful in resolving the initial issue I was grappling with.

However, during the process of implementing this solution, I encountered another challenge.

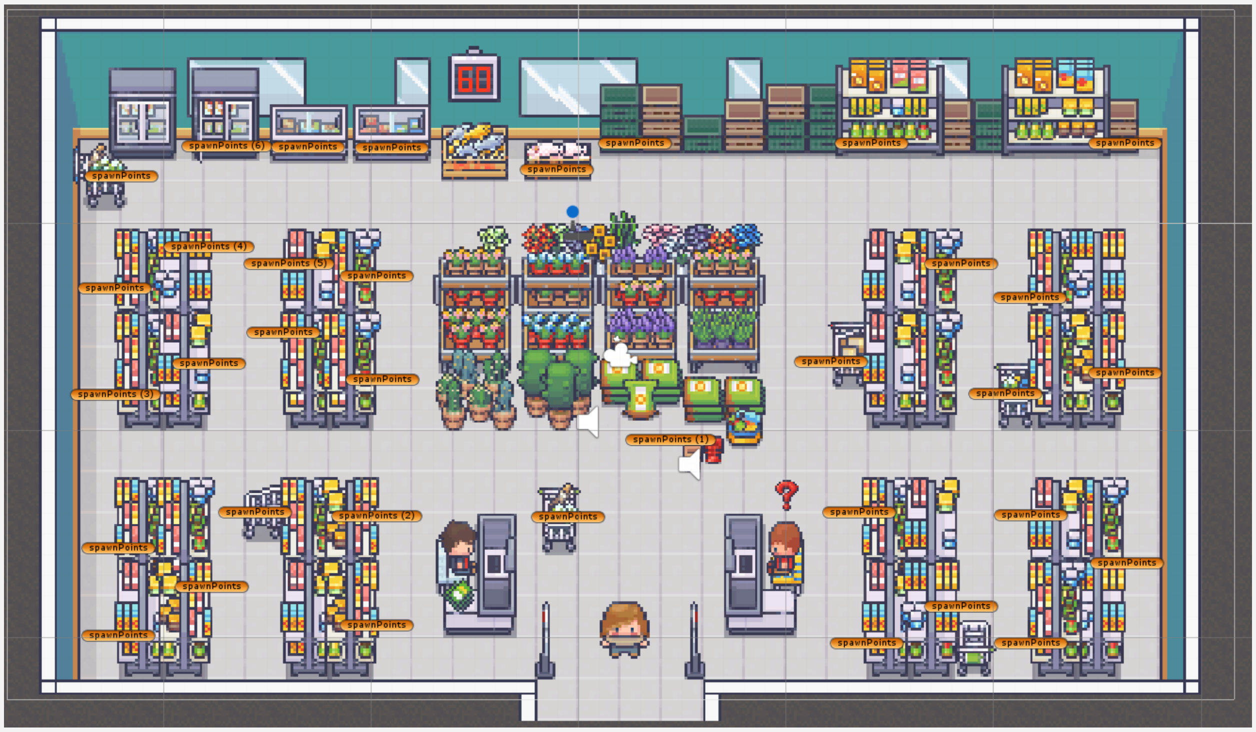
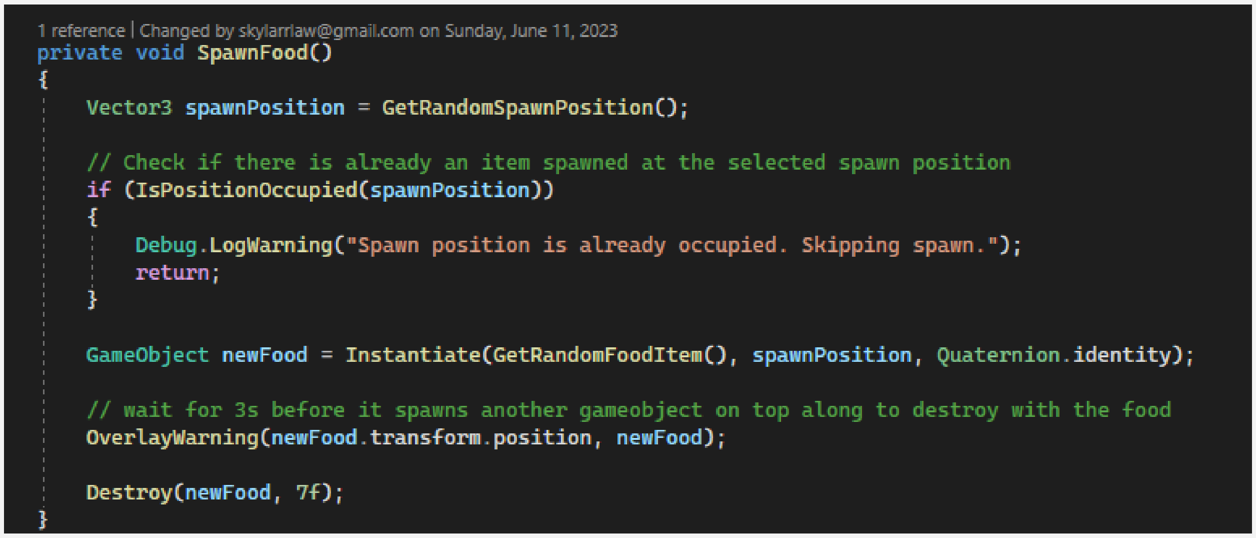


Figure 38: Predefined spawn locations

As I continued working on the spawner system, a new issue arose: occasionally, two pieces of food would unintentionally spawn at the exact same location. While this occurrence was rare, I had to decide whether to leave it to chance or find a way to address this probability mishap. Given the time remaining, I made the proactive choice to patch this issue and ensure a smoother gameplay experience. With determination, I delved into finding a solution that would eliminate the chance of simultaneous spawns at the same location.

To tackle this issue, I devised a simple yet effective solution. Before spawning a new item, I implemented a check to verify if the intended spawn position was already occupied. If the position was found to be occupied by another item, the spawner would skip that particular spawn and move on to the next available position. This approach ensured that no two items would inadvertently spawn at the same location, resolving the probability mishap and enhancing the overall gameplay experience. By implementing this check, I successfully mitigated the rare occurrence of simultaneous spawns at the exact same position, ensuring a smoother and more reliable gameplay flow.



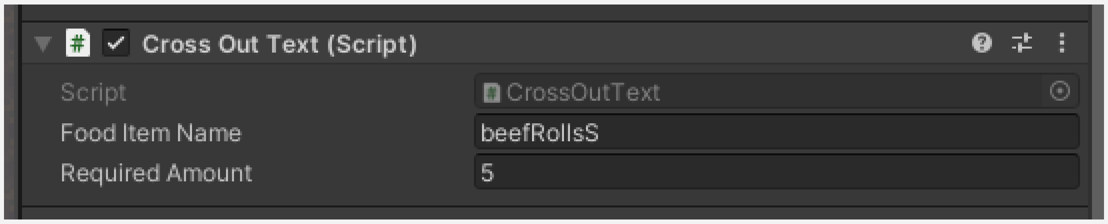
```
1 reference | Changed by skylarrlaw@gmail.com on Sunday, June 11, 2023
private void SpawnFood()
{
    Vector3 spawnPosition = GetRandomSpawnPosition();

    // Check if there is already an item spawned at the selected spawn position
    if (IsPositionOccupied(spawnPosition))
    {
        Debug.LogWarning("Spawn position is already occupied. Skipping spawn.");
        return;
    }

    GameObject newFood = Instantiate(GetRandomFoodItem(), spawnPosition, Quaternion.identity);

    // wait for 3s before it spawns another gameobject on top along to destroy with the food
    OverlayWarning(newFood.transform.position, newFood);

    Destroy(newFood, 7f);
}
```

Figure 39: Check spawn location is occupied

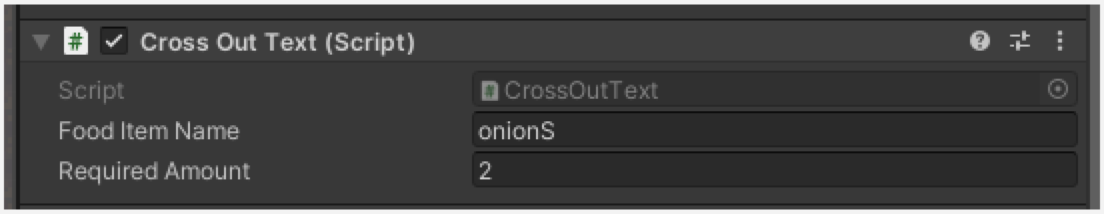
Following the implementation of the spawn position check, a minor issue surfaced. When a spawn location was found to be occupied, the corresponding food item would be skipped without being respawned elsewhere. While this was a small issue, it was brought to my attention during a discussion with David. He offered a plan to address this issue. However, considering the limited time available and my other pressing assignments, I made the decision not to pursue a fix for this particular issue. Although it was not a game-breaking problem and had minimal impact on the overall gameplay experience, I acknowledged its presence and chose to prioritise my time accordingly, perhaps this could be the first change to be made if I were to revisit the game again.

Once I had set up the food spawner, it was time to create a grocery list feature for the game. I envisioned an integrated grocery list that would dynamically cross out items as the player collected them. To implement this functionality, I sought assistance from my friend Vlad, who had previously implemented a similar feature in his own game.

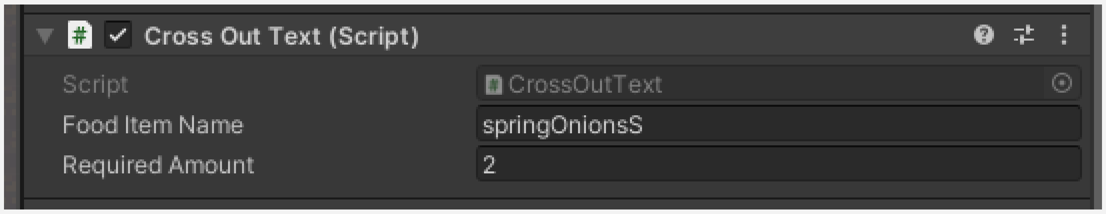
Integrating the grocery list with the player's inventory system required an additional step. I needed to establish a connection between my inventory system and the cross-out text script. Whenever the player picked up an item, the script would check if the item had met the threshold requirement and determine if it should be crossed out. To streamline the process, I added the functionality to individually specify the required quantity for each food item. This allowed me to easily adjust the amount needed without constantly modifying the script back and forth.



Script	CrossOutText
Food Item Name	beefRollsS
Required Amount	5

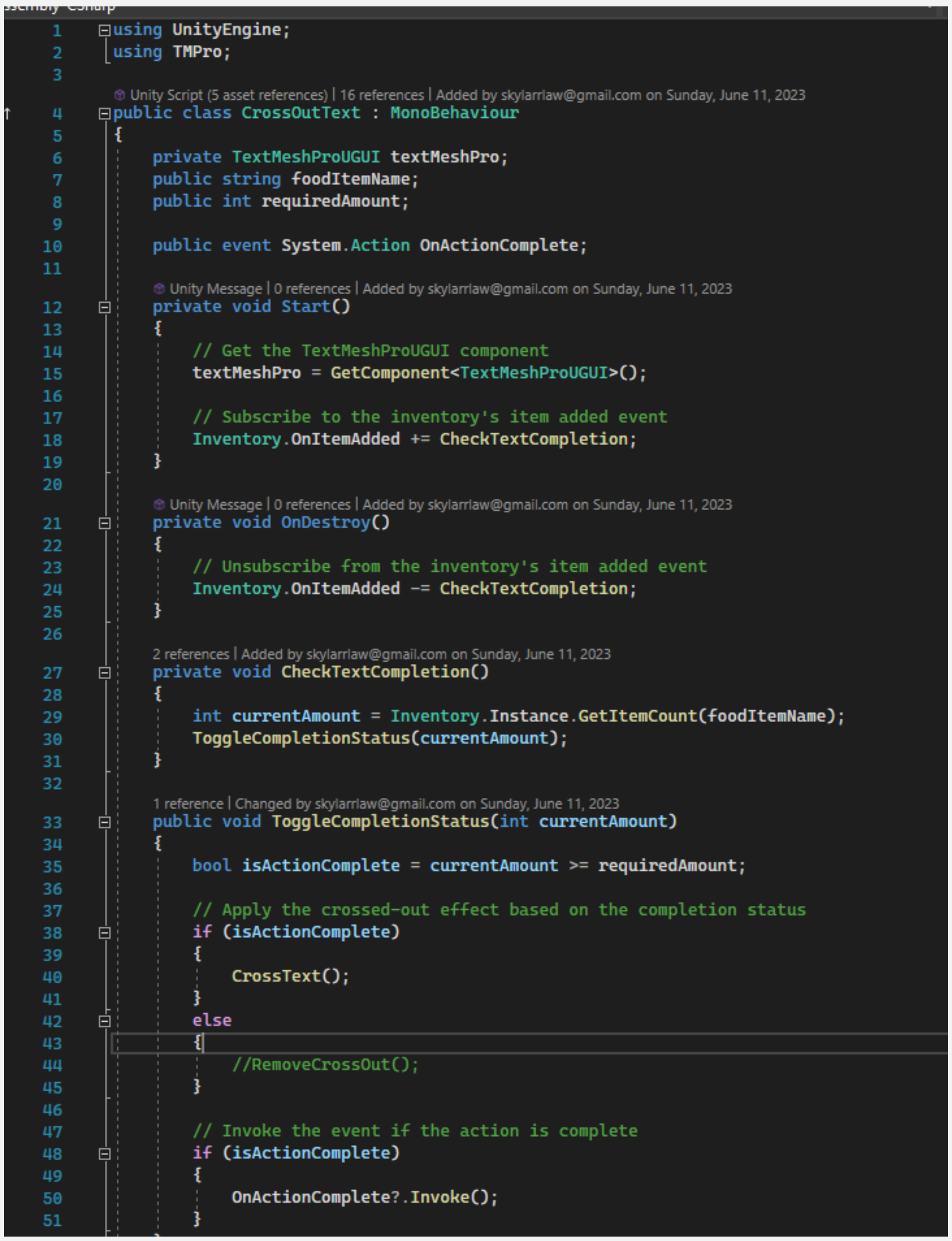


Script	CrossOutText
Food Item Name	onionS
Required Amount	2



Script	CrossOutText
Food Item Name	springOnionsS
Required Amount	2

Figure 40-42: Crossout text requirement check in inspector



```
using UnityEngine;
using TMPro;

Unity Script (5 asset references) | 16 references | Added by skylarrlaw@gmail.com on Sunday, June 11, 2023
public class CrossOutText : MonoBehaviour
{
    private TextMeshProUGUI textMeshPro;
    public string foodItemName;
    public int requiredAmount;

    public event System.Action OnActionComplete;

    Unity Message | 0 references | Added by skylarrlaw@gmail.com on Sunday, June 11, 2023
    private void Start()
    {
        // Get the TextMeshProUGUI component
        textMeshPro = GetComponent<TextMeshProUGUI>();

        // Subscribe to the inventory's item added event
        Inventory.OnItemAdded += CheckTextCompletion;
    }

    Unity Message | 0 references | Added by skylarrlaw@gmail.com on Sunday, June 11, 2023
    private void OnDestroy()
    {
        // Unsubscribe from the inventory's item added event
        Inventory.OnItemAdded -= CheckTextCompletion;
    }

    2 references | Added by skylarrlaw@gmail.com on Sunday, June 11, 2023
    private void CheckTextCompletion()
    {
        int currentAmount = Inventory.Instance.GetItemCount(foodItemName);
        ToggleCompletionStatus(currentAmount);
    }

    1 reference | Changed by skylarrlaw@gmail.com on Sunday, June 11, 2023
    public void ToggleCompletionStatus(int currentAmount)
    {
        bool isActionComplete = currentAmount >= requiredAmount;

        // Apply the crossed-out effect based on the completion status
        if (isActionComplete)
        {
            CrossText();
        }
        else
        {
            //RemoveCrossOut();
        }

        // Invoke the event if the action is complete
        if (isActionComplete)
        {
            OnActionComplete?.Invoke();
        }
```

Figure 43: Crossout text script

With the grocery list feature successfully implemented, I finally had a moment to catch my breath. The past two weeks had been intense, leaving little time for rest or to focus on anything else. So, I took a moment to relax and contemplate the next steps for the game.

During this reflective period, I decided to delve into the possibilities of narrative branching and multiple endings. I wanted to add depth and replayability to the game by offering players different paths and outcomes based on their choices and actions. This narrative branching approach intrigued me, as it would allow players to have a unique and personalised experience each time they played. Exploring this narrative aspect would require careful planning and the creation of various storylines and decision points. I knew it would be challenging, but the potential to create a captivating and immersive game experience motivated me to pursue this route.

As I continued brainstorming, a realisation struck me: I already had a functional inventory and crossout system in place. This presented an opportunity to expand upon it and create a unique ending screen that reflected the player's choices and actions throughout the game. I envisioned an ending screen where each food item the player collected or failed to collect would be represented by a corresponding sprite of food. This would serve as a visual representation of their progress and the impact of their decisions. For example, if the player successfully gathered all the required groceries, the ending screen would display a complete dish. On the other hand, if the player missed certain items, the dish would be incomplete.

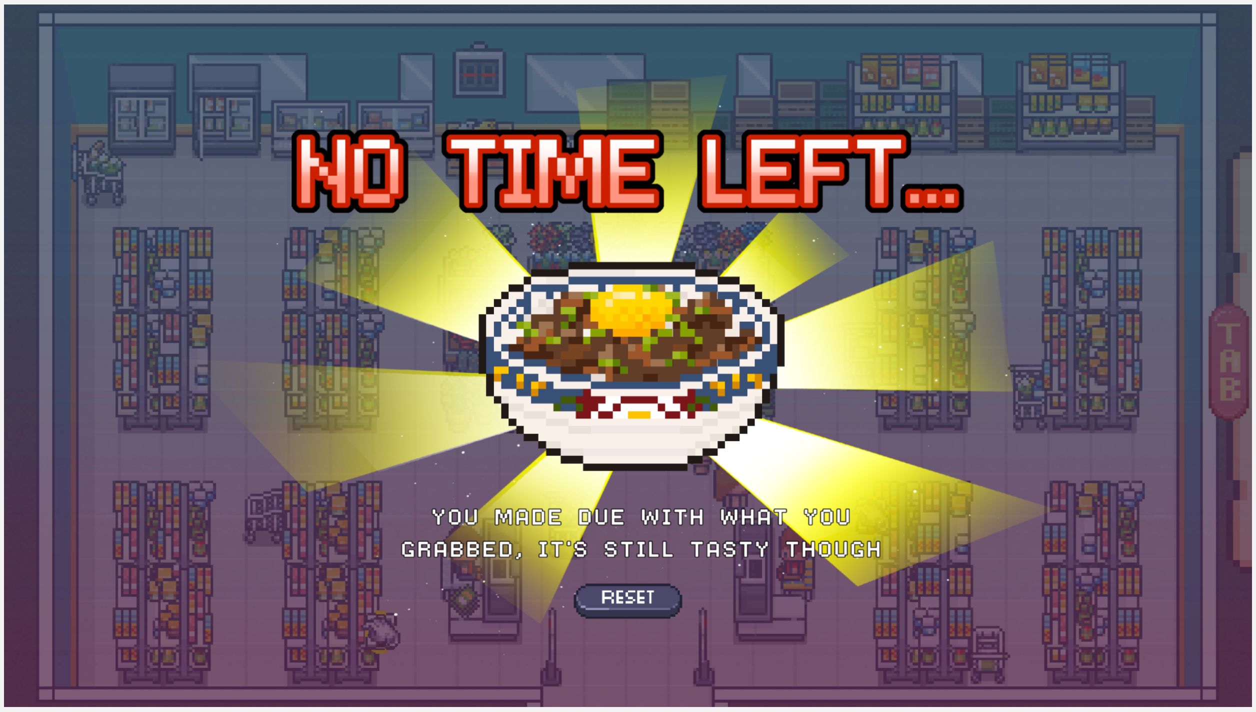


Figure 44: Branching ending based on the food you collected

This concept excited me, as it added another layer of visual storytelling and personalization to the game. It would give players a tangible representation of their accomplishments and choices, reinforcing the sense of agency and immersion. With this idea in mind, I eagerly set out to implement the ending screen, creating unique visuals for each food item and programming the logic to display the appropriate sprites based on the player's inventory status. I was thrilled by the potential of this feature to enhance the overall experience and create a memorable conclusion to the game.

To implement the functionality of enabling and disabling the sprites of the food items based on the player's progress, I designed a system where all the sprites of the required food items were initially disabled. Using a script, I enabled specific sprites corresponding to the food items that the player had collected during gameplay. This allowed for dynamic updating of the sprites based on the player's progress. I initially encountered a challenge where the sprites would not disable even if the corresponding food had not been collected. However, through careful investigation, I identified the root cause of the problem. Instead of enabling or disabling the entire canvas containing the food sprites, I realised that I needed to specifically target the individual panels and sprites within the canvas. By making this adjustment and calling the relevant panels and sprites, I successfully resolved the issue and achieved the desired functionality of displaying only the collected food items on the ending screen. With this, my core gameplay was pretty much complete.

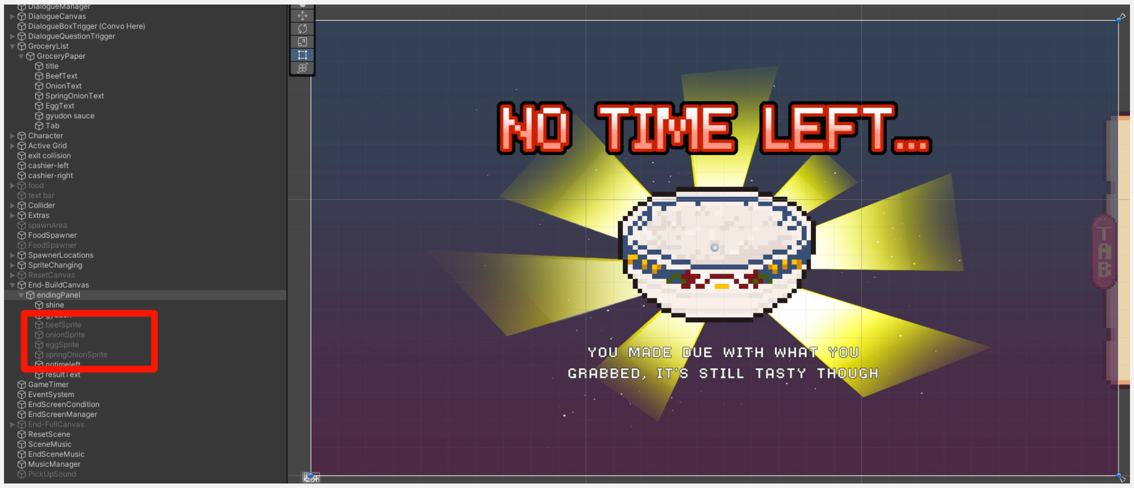
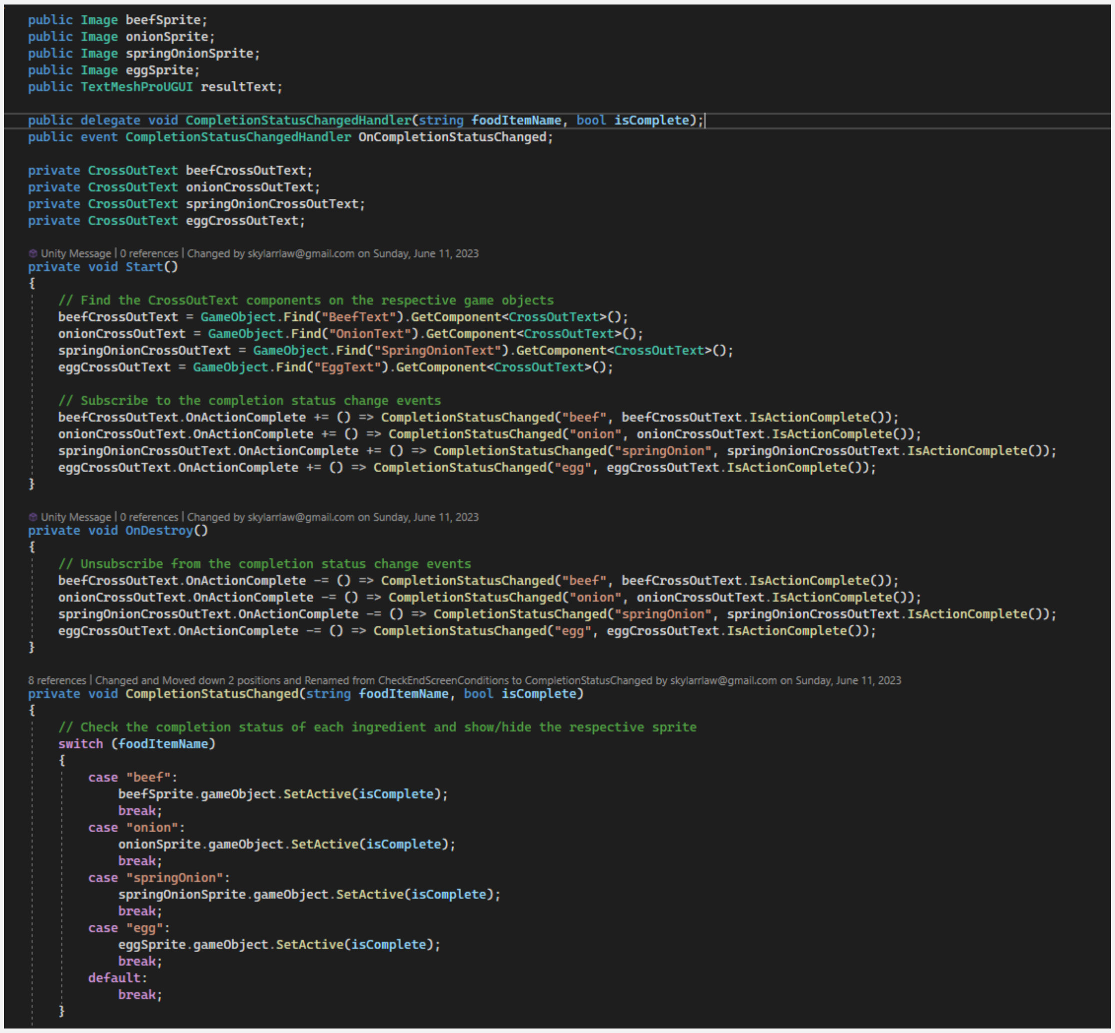
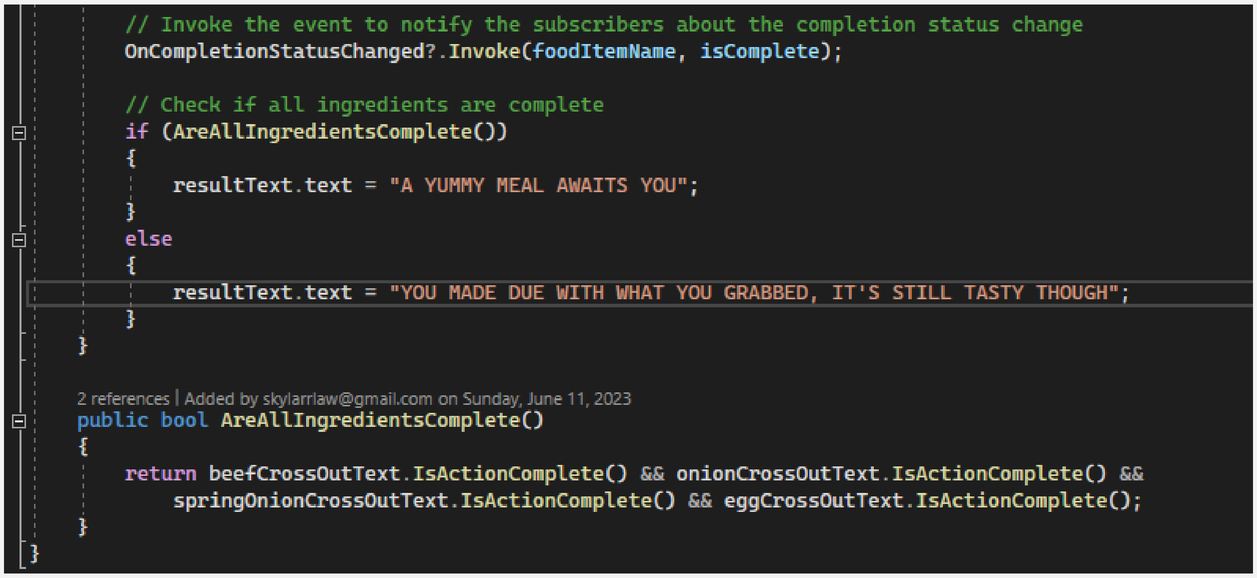


Figure 45: Food sprites initially disabled



```
public Image beefSprite;
public Image onionSprite;
public Image springOnionSprite;
public Image eggSprite;
public TextMeshProUGUI resultText;

public delegate void CompletionStatusChangedHandler(string foodItemName, bool isComplete);
public event CompletionStatusChangedHandler OnCompletionStatusChanged;

private CrossOutText beefCrossOutText;
private CrossOutText onionCrossOutText;
private CrossOutText springOnionCrossOutText;
private CrossOutText eggCrossOutText;

Unity Message | 0 references | Changed by skylarrlaw@gmail.com on Sunday, June 11, 2023
private void Start()
{
    // Find the CrossOutText components on the respective game objects
    beefCrossOutText = GameObject.Find("BeefText").GetComponent<CrossOutText>();
    onionCrossOutText = GameObject.Find("OnionText").GetComponent<CrossOutText>();
    springOnionCrossOutText = GameObject.Find("SpringOnionText").GetComponent<CrossOutText>();
    eggCrossOutText = GameObject.Find("EggText").GetComponent<CrossOutText>();

    // Subscribe to the completion status change events
    beefCrossOutText.OnActionComplete += () => CompletionStatusChanged("beef", beefCrossOutText.IsActionComplete());
    onionCrossOutText.OnActionComplete += () => CompletionStatusChanged("onion", onionCrossOutText.IsActionComplete());
    springOnionCrossOutText.OnActionComplete += () => CompletionStatusChanged("springOnion", springOnionCrossOutText.IsActionComplete());
    eggCrossOutText.OnActionComplete += () => CompletionStatusChanged("egg", eggCrossOutText.IsActionComplete());
}

Unity Message | 0 references | Changed by skylarrlaw@gmail.com on Sunday, June 11, 2023
private void OnDestroy()
{
    // Unsubscribe from the completion status change events
    beefCrossOutText.OnActionComplete -= () => CompletionStatusChanged("beef", beefCrossOutText.IsActionComplete());
    onionCrossOutText.OnActionComplete -= () => CompletionStatusChanged("onion", onionCrossOutText.IsActionComplete());
    springOnionCrossOutText.OnActionComplete -= () => CompletionStatusChanged("springOnion", springOnionCrossOutText.IsActionComplete());
    eggCrossOutText.OnActionComplete -= () => CompletionStatusChanged("egg", eggCrossOutText.IsActionComplete());
}

8 references | Changed and Moved down 2 positions and Renamed from CheckEndScreenConditions to CompletionStatusChanged by skylarrlaw@gmail.com on Sunday, June 11, 2023
private void CompletionStatusChanged(string foodItemName, bool isComplete)
{
    // Check the completion status of each ingredient and show/hide the respective sprite
    switch (foodItemName)
    {
        case "beef":
            beefSprite.gameObject.SetActive(isComplete);
            break;
        case "onion":
            onionSprite.gameObject.SetActive(isComplete);
            break;
        case "springOnion":
            springOnionSprite.gameObject.SetActive(isComplete);
            break;
        case "egg":
            eggSprite.gameObject.SetActive(isComplete);
            break;
        default:
            break;
    }
```

Figure 46: Ending scene manager script



```
        // Invoke the event to notify the subscribers about the completion status change
        OnCompletionStatusChanged?.Invoke(foodItemName, isComplete);

        // Check if all ingredients are complete
        if (AreAllIngredientsComplete())
        {
            resultText.text = "A YUMMY MEAL AWAITS YOU";
        }
        else
        {
            resultText.text = "YOU MADE DUE WITH WHAT YOU GRABBED, IT'S STILL TASTY THOUGH";
        }
    }

    2 references | Added by skylarrlaw@gmail.com on Sunday, June 11, 2023
    public bool AreAllIngredientsComplete()
    {
        return beefCrossOutText.IsActionComplete() && onionCrossOutText.IsActionComplete() &&
            springOnionCrossOutText.IsActionComplete() && eggCrossOutText.IsActionComplete();
    }
}
```

Figure 47: Text change depending on whether you grabbed all the items

To wrap up the game, I implemented a main menu to provide a polished presentation. Additionally, I incorporated background music and sound effects to enhance the overall immersion during gameplay.

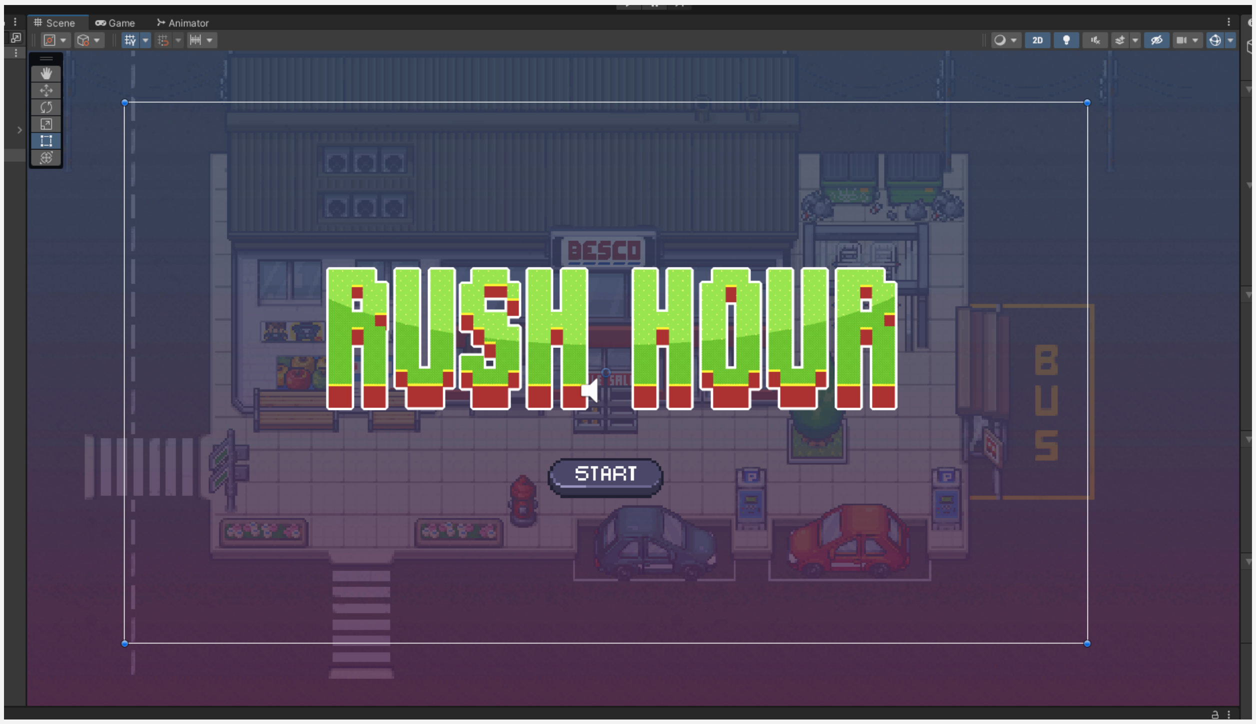
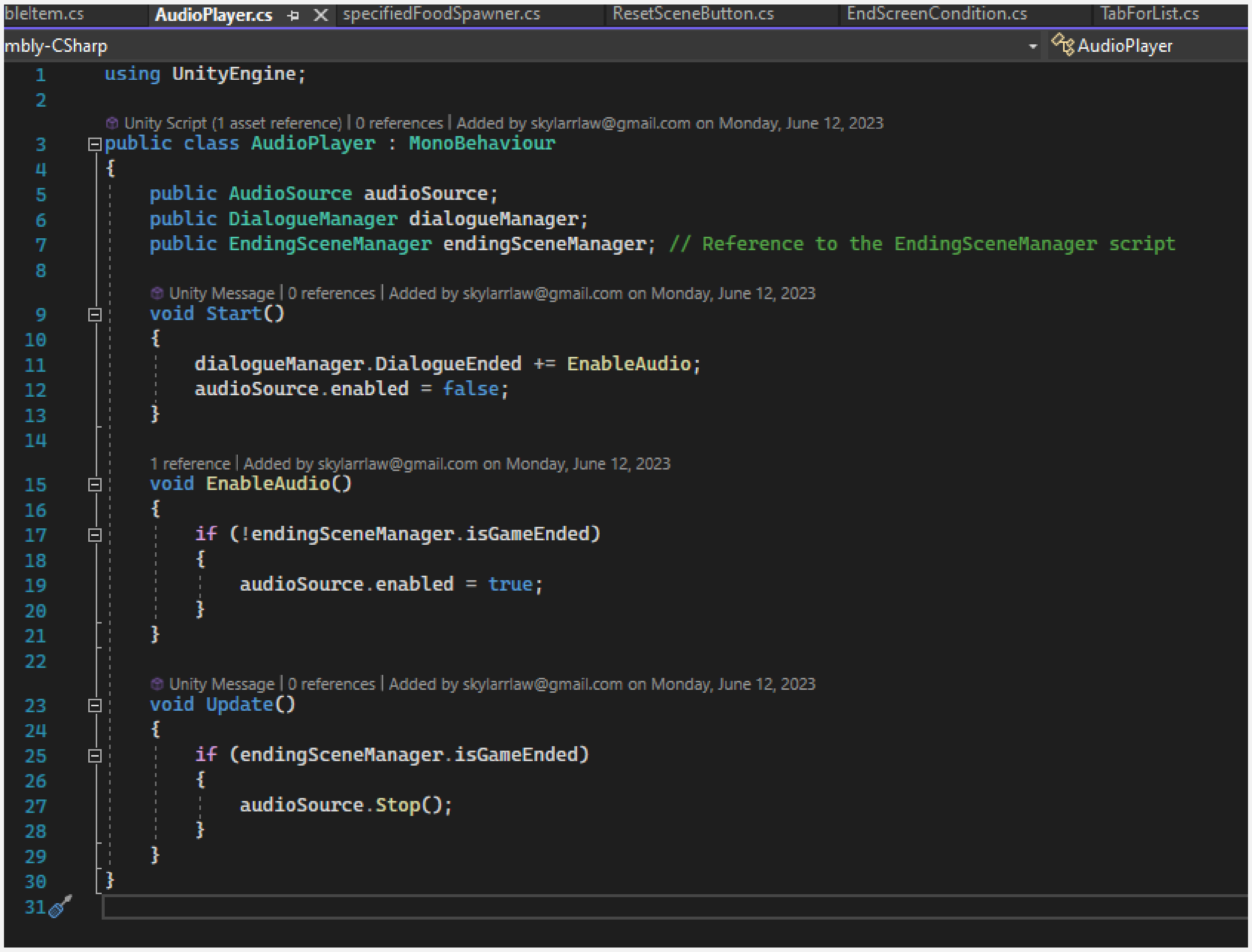


Figure 48: Rush Hour main menu

To manage the audio effectively, I developed a script that checks specific conditions, such as the completion of dialogue or the game itself, to control the playback of music. This implementation worked seamlessly, ensuring that the music stops appropriately at the desired moments, and ultimately helped me finalise the game.



```
using UnityEngine;

Unity Script (1 asset reference) | 0 references | Added by skylarrlaw@gmail.com on Monday, June 12, 2023
public class AudioPlayer : MonoBehaviour
{
    public AudioSource audioSource;
    public DialogueManager dialogueManager;
    public EndingSceneManager endingSceneManager; // Reference to the EndingSceneManager script

    Unity Message | 0 references | Added by skylarrlaw@gmail.com on Monday, June 12, 2023
    void Start()
    {
        dialogueManager.DialogueEnded += EnableAudio;
        audioSource.enabled = false;
    }

    1 reference | Added by skylarrlaw@gmail.com on Monday, June 12, 2023
    void EnableAudio()
    {
        if (!endingSceneManager.isGameEnded)
        {
            audioSource.enabled = true;
        }
    }

    Unity Message | 0 references | Added by skylarrlaw@gmail.com on Monday, June 12, 2023
    void Update()
    {
        if (endingSceneManager.isGameEnded)
        {
            audioSource.Stop();
        }
    }
}
```

Figure 49: Rush hour music system

8. Playtesting & Showcases

Over the past few weeks, I spent a lot of time having work sessions with Vlad, Kathi and Anu. Most of them spent a lot of time reviewing my progress of the game via Discord screen share. I decided to add their feedback in this section as part of showcases.

8.1. Playtest Session - 5/6/2023

Overall Feedback/Changes

	Feedback/Observations	Changes	Reason/Thoughts
1	Players feels uncertain about the game's message, the	I opted for a complete overhaul of the game's aesthetics,	Initially, the visual style of the game was designed

	aesthetics of the game does not fit the message intended	transforming it into a vibrant and colourful experience. To achieve this, I incorporated an asset pack created by LimeZu from itch.io, which replaced the original gloomy and monochromatic 1-bit black and white style. This change injected new life and visual appeal into the game, enhancing the overall atmosphere and creating a more engaging and visually stimulating environment for players to enjoy.	specifically for the Playdate, a device that can only display black or white/transparent pixels. During the early stages, aesthetics were not the primary focus as my main goal was to create a functional prototype that showcased the core mechanics of the game.
2	Players were unsure about what the game mode is about	Food spawner and general aesthetics change was made.	At the time, the game had a deficiency in mechanics and a lack of coherence between aesthetics and the topic. Due to time constraints, significant progress was challenging to achieve as the changes were only initiated a couple days ago.
3	Observations revealed that the visuals looked more like a horror game than a slice of life themed game	The aesthetics have been transformed into a vibrant and location-appropriate style, specifically resembling a grocery store setting.	The black and white theme gave the game a gloomy and horror-like appearance, which was not aligned with the intended slice of life genre.
4	Players felt that the "mushroom" object as a powerup can be misinterpreted as another food to collect as their objective. Players also felt that the purpose of powerup can be missed due to the misinterpretation.	Initially, power-ups were replaced with stars in the game, but ultimately, they were removed in the final version.	The mushroom sprite was temporarily used as a placeholder during testing.

8.2. Showcase Session - 9-11/6/2023

Overall Feedback/Changes

	Feedback/Observations	Changes	Reason/Thoughts
1	Observers expressed the opinion that the game's duration was insufficient, leaving players with limited opportunities to seize any items.	To address the issue of players feeling rushed and to allow them ample time to process the grocery list requirements and familiarise themselves with the game controls, I decided to extend the available time in the gameplay. This adjustment aimed to provide a more relaxed and comfortable experience, enabling players to better strategise and accomplish their grocery list objectives while also becoming proficient with the necessary key navigation.	I overlooked the fact that some players might not be accustomed to playing fast-paced games or be familiar with common key functions such as WASD and TAB. Consequently, I failed to consider the impact of the short duration on their gameplay experience.
2	In the older iterations, observers felt that having multiple waves makes the game feel too repetitive.	I made the decision to completely remove the wave structure from the game. Instead, I adjusted the game's duration to allow for a longer timeframe, during which all the ingredients would be available to collect simultaneously.	During the brainstorming phase, I believed that incorporating multiple waves with increasing difficulties would elevate the stress factor in the game. However, I overlooked the fact that this approach could also result in excessive repetition, potentially diminishing the overall gameplay experience.
3	Observers noted that the beef rolls were not appearing frequently enough, leading to players consistently failing to acquire the beef.	In order to tackle the issue of infrequent beef roll spawns, I made the decision to broaden the selection of food items that could appear. Specifically, I included an additional beef roll in the list of items for the food spawner to choose from. This modification aimed to increase the likelihood of beef rolls being spawned more frequently relative to the other food items, ultimately enhancing the chances of obtaining beef during	As the game transitioned from the wave structure to a longer timeframe and included all the food items in the food spawner's list, an unintended consequence emerged. The food spawner no longer prioritised specific food items, leading to a decrease in the frequency of beef roll spawns. This presented a challenge for players as beef rolls were both crucial and required in larger quantities to complete

		gameplay.	their grocery lists. Consequently, players encountered difficulties in obtaining an adequate number of beef rolls by the end of the game.
4	Observers realised that there was no indicator to speak to the cashier for instructions.	To provide visual cues for interaction, I introduced an animation featuring a question mark and the letter "E" above the cashier whom players were supposed to speak to. This addition aimed to assist players in identifying the specific character with whom they needed to interact.	The indicator was not given a high priority during development. However, it was subsequently added at a later stage to address the need for clearer visual guidance or cues in the game.
5	Observers expressed uncertainty regarding which key to press in order to advance the dialogue.	To address the issue of uncertainty, I implemented an animation of the "E" key in the bottom right corner of each dialogue cue. This visual indicator served to clarify that the "E" key was the designated input for advancing the dialogue.	To avoid using a large block of text stating "Press E to continue," I sought alternative methods to visually indicate the continuation of dialogue. Additionally, I initially assumed that players would instinctively press the "E" key to interact or proceed. However, I came to realise that this assumption was not accurate, as not all players are accustomed to games that utilise the "E" key for interaction.
6	Observers had a negative perception of the collisions, which were mapped according to the sprite shapes, as they were deemed highly erratic and frequently caused the player's character to become stuck.	I opted to disable collisions on the sprites and instead implemented separate square boxes to handle the collision. This decision grants me the flexibility to meticulously design collision shapes for specific areas, tailoring them to fit the desired gameplay experience.	Originally, I was unaware of alternative methods for implementing collisions aside from using the original sprite's collisions. However, thanks to Louis's suggestion, I realised that there was a better approach. Louis explained that utilising simple square collision boxes would improve the game's performance

			compared to custom-shaped collisions based on the sprites. This insight prompted me to make the necessary changes and optimise the game's collision system accordingly.

8.3. Playtest Session - 12/6/2023

Overall Feedback/Changes

	Feedback/Observations	Changes	Reason/Thoughts
1	Players felt that the collisions of the aisles and trolleys feel very janky as it blocks the pathway of the player character. It can be hard for the players to move around the environment and sometimes get stuck.	Collisions were then changed to have curved edges. It allows players to glide through certain corners better.	Collision boxes initially were derived from the actual sprites. Louis suggested that I should create invisible boxes which allows me to handle the collision easier. At the time, I was not aware that collision boxes could be curved via the edge radius option in the Box Collider 2D inspector window. Knowing that, I immediately added the option.
2	Observations revealed that the game felt very silent.	Background music was added.	Background music was always meant to be added, it was not added in time for this playtest session.
3	Players felt that it lacked visual or audio confirmation of item pickups. Sometimes, players are unsure if they had picked up the item they were grabbing.	Sound effects for item pickup were added.	Initially, I contemplated incorporating either a visual or audio response. Due to time constraints, I was unable to create a visual response and instead chose to implement an audio response. Unfortunately, the audio response was not included in time for the playtest session. However, towards the end of the day, I managed to showcase the newly added

			changes to David.
4	Players felt that it lacked accessibility options, it can be hard for players to remember what the interaction buttons are with so much going on in the game.	To enhance usability, I included a "TAB" key sprite beside the grocery list slideout sprite, complementing the dialogue that informed users about using the TAB key to access the grocery list.	At the beginning, I believed that including the dialogue mentioning the TAB key would be sufficient. However, I later realised that the high-stress nature of the game could cause players to forget the information provided in the dialogue as well.
5	Observations revealed that it is lacking a main menu screen. Throwing players into the game itself confuses some players as their motive for the game is uncertain.	Main menu was added to make the game feel more complete.	Creating the main menu was not a priority and was placed lower on my list of tasks. However, it was eventually added to the game to provide a more polished and complete experience for the players.
6	Players do feel that it is fun and exciting to play, it achieved the stress factor that the game was meant to represent	No changes were made.	I'm delighted that the changes made had a positive impact on the players, as it indicates that the intended goals were successfully accomplished.

8.4. Brief Reflection

I highly valued the feedback I received, even though the sample size was relatively small due to the slower progress I made during the game development. The quality of the feedback was more important than the quantity, as it provided invaluable insights that greatly influenced the game's outcome. Without the feedback, I wouldn't have been able to push the game to its current extent and improve it accordingly.

9. Critical Reflection

Although I had set high aims for myself this term, I realised that the initial goals I had set were not fully met. My intention was to create a Playdate console game, which would allow me to explore other platforms and expand my knowledge of game development. While I wasn't able to create the exact game I had envisioned, the learning progress I made over these few weeks was still valuable and worthwhile. The knowledge I gained can be applied to future projects.

Lua, being a powerful programming language, shares similarities with C# that we are currently using. It is widely used in games like Roblox due to its easy learning curve, making it accessible even for younger kids. However, I underestimated the challenges posed by the closed SDK library system of Playdate, as well as the implementation of the LDTK system and libraries.

On the narrative and slice of life elements, I believe I constructed them well and successfully implemented a small-scale branching narrative in the game. The feedback from the second playtest session on June 12th was positive, with players expressing that they could feel the intended message and stress factor in the game. This reassured me that the emotions I aimed to convey were successful.

Looking ahead, I plan to conduct thorough research and allocate myself a more realistic timeframe for future projects. I've learned that having a scope that is too large can lead to increased stress and unmet expectations. By properly managing my expectations and giving myself adequate time, I can enhance the overall development process and achieve better results.

10. Credits

Assets used for Development

Visuals

Pixel Art Interior Sprites: https://limezu.itch.io/moderninteriors
Pixel Art Exterior Sprites: https://limezu.itch.io/modernexteriors
Pixel Art UI Sprites (Dialogue Box, Buttons, Grocery List): https://soulares.itch.io/free-ui
Pixel Art styled Font: https://grafxkid.itch.io/at01

Audio

Retro RPG Music Pack (Battle Won, Fight 1, Curious, Festive):
https://cat2x.itch.io/retro-rpg-music-pack
1Bit CanariPack SFX (Jump01): https://canarigames.itch.io/canaripack-1bit-topdown