Project Design

Solution Overview:
The Legend of Calculus is a 2D, open world game in which players can explore a fantasy world and complete quests for historical mathematicians by solving calculus problems. The game encourages players to practice calculus concepts and be inspired to continue learning.

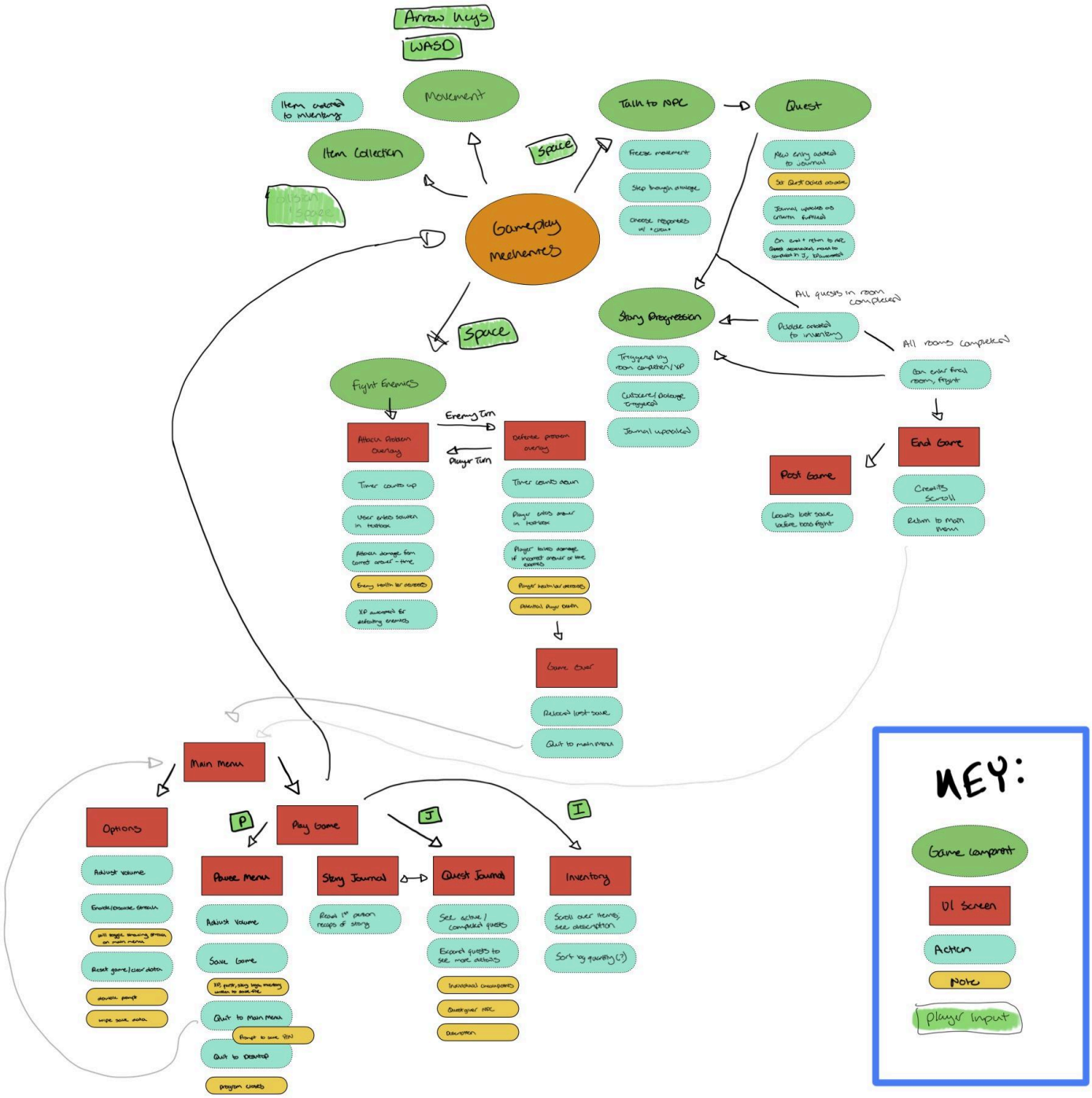
Record of Tasks:

Task Number	Planned Action	Planned Outcome	Time Estimation	Target Completion Date
1	Thinking of ideas for a final project.	Coming up with a list of several plausible ideas for a final project.	1 class	January 30
2	Considering each of my ideas and the skills I already have and would need to learn to implement it.	Decide on a final project idea that meets my skill level but would still challenge me.	1 class	February 1
3	Describe the vision for my final project idea, determine who my client would be and what resources I would need.	Have a clearer vision of my project and what I need to start working on it.	1 class	February 5
4	Reflection meeting on vision for final project.	Receive and give feedback and potential ideas from classmates.	1 class	February 9
5	Interview client for the first time	Get an idea of what client would like to see in game product and story	1 classes	February 15
6	Use client feedback in order to refine vision for game and creating a rationale for the product and success criteria.	Finish Final Planning Document	2 classes	February 16
7	Update website to include information from planning unit.	Website published	1 class	February 16
8	Sketch out how user would interact with game product and move between screens/mechanics.	Finish design flowchart and project overview on Design document.	1 class	February 22
9	Create UML diagrams in order to plan custom classes that will be used in project.	Finish Data Table in Design document.	1 classe	February 24
10	Create a test plan for different mechanics in the game product.	Finish Test Plan in Design document.	1 class	February 24
11	Think about how arrays, loops, and data structures will be used in game project.	Finish Design Complexity in Design document.	1 class	March 1

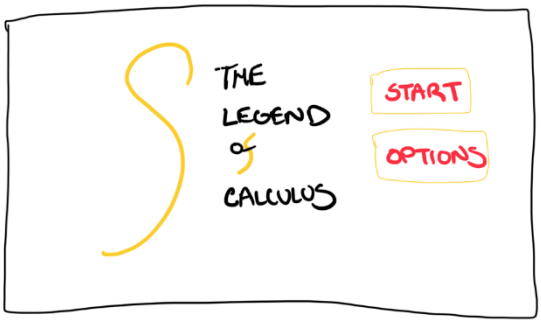
12	Pseudocode various methods for game product logic managers like Dialogue, Inventory, Quest, and Journal that will be handling data.	Finish Algorithms section in Design document.	2 classes	March 7
13	Create a plan mapping previously identified success criteria to actions, outcomes, and dates.	Finish Record of Tasks section in Design document.	1 class	March 8
14	Sketch out imagined displays and screens for game product.	Finish Screen Prototypes section in Design document.	2 classes	March 9
15	Update website to include design information.	Website published with Unit 6 documents.	1 day	March 10
16	Create a player sprite; start coding Player script.	Player can move and cycle through animations.	1 class	March 11
17	Create Enemy sprites; start coding Enemy script	Player collision with Enemy pauses game world and opens new overlay.	1 class	March 13
18	Code Python script to generate calculus problems OR create a database of calculus problems to randomly select from. Integrate it with Enemy combat script.	Player can defeat an enemy by solving a calculus problem.	3 classes	March 21
19	Create Settings overlay and code Settings UI script.	Player can toggle Pause menu open/closed by pressing “S”. From Settings menu can quit game.	1 class	April 3
20	Code InventoryManager and Item script	Player can collide with items in game world to collect them.	2 class	April 5
21	Create a Character sprite; start coding Character script. Create DialogueManager.	Player can interact with Character NPC and step through dialogue.	1 class	April 9
22	Create QuestManager	Player can start a quest by talking to an NPC and finish the quest by completing quest criteria.	2 classes	April 16
23	Create Journal UI; code JournalManager	Player can toggle Journal page open/closed by pressing “J”. Quests populate journal as they are started	2 classes	April 22
24	Find tileset and draw maps.	Map for main room and 1 other room is drawn and imported with tilemap colliders as appropriate. Player can walk around map.	1 class	April 24
25	Draw Player sprite and animate	Player has final appearance and animations.	1 class	April 26
26	Populate finished room with quests.	Player can complete all quests in room.	2 classes	May 2

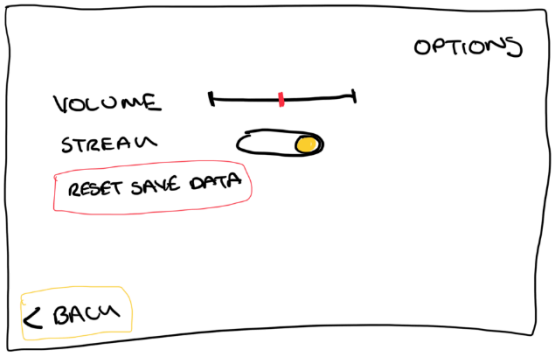
27	Code Player script that will track when to trigger cutscene/maybe create terminal in main room.	Finishing a room triggers a story beat/cutscene.	1 class	May 7
28	Create start screen with options page.	Launching game launches to main menu/start screen. From there new game can be started.	1 class	May 9
29	Saving game populates save file with Quest, Inventory, Journal, Player, and Story stats. Restarting game loads player into main room with all that intact.	Player can start game, save & quit game from pause menu, and restart game with progress intact.	1 class	May 13
30	Game demo published on website.	Website supports game download OR web embedded functionality.	1 day	May 14

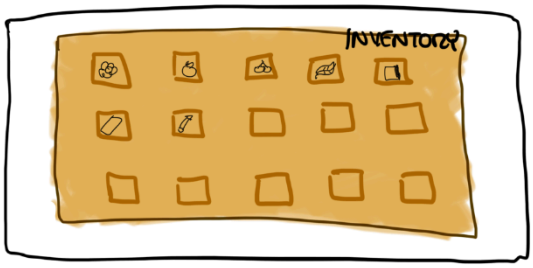
Flowchart:

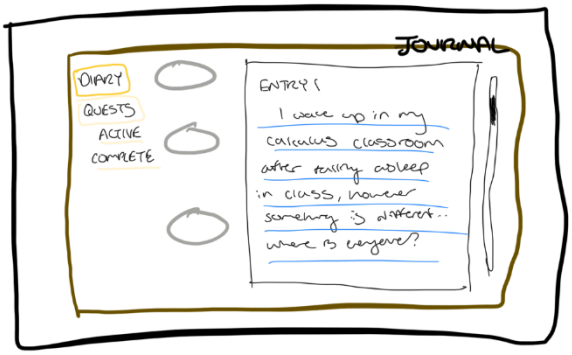

Arrow keys
WASD
Movement
Item added to inventory
Item Collection
Hidden Message
Space
Talk to NPC
Quest
Freeze movement
Step through dialogue
Choose responses w/ ↑ ↓ enter
New entry added to journal
1st Quest added as active
Journal updated as current objective
On exit + return to NPC dialogue changed, move to another ? if necessary
Gameplay Mechanics
Space
All quests in room completed
Story Progression
Puzzle created to inventory
All rooms completed
Triggered by room completion/XP
Cutscene/ dialogue triggered
Journal updated
Can enter final room, fight
Fight Enemies
Enemy Turn
Player Turn
Attack Problem Overlay
Defense problem overlay
Timer counts up
User enters answer in textbox
Player takes damage if incorrect answer or time expires
Enemy health bar decreases
XP awarded for defeating enemies
Timer counts down
Player enters answer in textbox
Player health bar decreases
Potential Player death
End Game
Post Game
Credits Scroll
Return to main menu
Locks last save before boss fight
Game Over
Restart last save
Quit to main menu
Main Menu
Options
Play Game
P
J
I
Adjust volume
Enable/disable controls
pull toggle streaming prompt on main menu
Reset game/clear data
Are you sure prompt
Wipe save data
Pause Menu
Story Journal
Quest Journal
Inventory
Adjust Volume
Save Game
XP, path, story keys, inventory written to save file
Quit to Main Menu
Prompt to save, Y/N
Quit to Desktop
Program closes
Read 1st person recaps of story
See active / completed quests
Expand quests to see more details
Individual checkpoints
Quest goal title
Description
Scroll over items, see description
Sort by quantity (?)
KEY:
Game component
UI Screen
Action
Note
Player Input

Screen Prototypes:









Data Table:

Item	Description	Data Type	Sample Data
Item	Stores information related to game world items that can be collected.	Custom **Item** Sprite spriteMask; Dictionary<> data; - Keys: name, quantity, description	spriteMask: <png pixel art apple> {data: name: “apple”, quantity: 5, description: “yummy fruit. Restores 5 health”}
Inventory	Manages player inventory.	Custom Class **Inventory** Private Item[] inventory public AddToInventory(Item item) Public RemoveFromInventory(string itemName) private SortInventory()	N/A- array of Items
Quest	Stores information related to a singular quest that can be completed.	Custom Class **Quest** String questName String[] questDialogue int currentDialogueIndex Dictionary<>[] questCriteria String[] journalEntries	name: questDialogue = [“Here is a quest. Would you like to begin?”, “The quest is to find the location of the inte-griffin and defeat it.”, “Have you found the griffin yet?”, “Thanks for defeating the griffin! Here is a reward!”] currentDialogueIndex = 0 questCriteria = {“location-found”: true, “griffin-defeated: false} journalEntries = [“quest begun entry”, “criteria update entry”, “quest complete entry”]
QuestManager	Manages different quest states.	Quest[] quests int[] activeQuests int[] completedQuests StartQuest(int questIndex) UpdateQuestCriteria(int questIndex, String criteriaKey) EndQuest(int questIndex) UpdateQuestJournal(int	quests = [Quest(), Quest(), Quest()] activeQuests = [0, 2] completedQuests = [1]

		questIndex)	
JournalEntry	Stores information related to a journal entry.	String title int linkedQuestIndex String currentText String[] criteriaBulletPoints int[] completedCriteriaPoints bool journalEntryComplete	title = “Discover The Forest of Fundamental Calculus linkedQuestIndex = 5 currentText = “The mysterious voice instructed you to find the forest.” criteriaBulletPoints = [“discoverForest”, “talk to someone in the forest”] completedCriteriaPoints = [0] journalEntryComplete = false
JournalManager	Manages backend of Journal UI	JournalEntry[] journalEntries int[] completeEntries UpdateJournalEntry()	journalEntries = [JournalEntry(), JournalEntry()] completeEntries = [1]
Location	Stores information related to a specific place that the player can discover.	String name int questRelation String questKeyRelation	name = “The Forest of Fundamental Calculus” questRelation = 5 questKeyRelation = “discover forest location”
Enemy	Stores information about an enemy that can attack player	Custom Class **Enemy** Sprite spriteMask int health float moveSpeed float timeForPlayerToSolveAttack TakeDamage() AttackPlayer()	spriteMask = <linked to enemy pixel art> health = 100 moveSpeed = 14 timeForPlayerToSolveAttack = 2
Character	Stores information about NPC characters that Player can talk to and receive quests from.	Custom Class **Character** Sprite spriteMask String name; String[] dialogue; int currentDialogueIndex; int[] questsToGive; Speak() OfferQuest(int questIndex)	spriteMask = <NPC pixel art> name = “Aeneas” dialogue = [“Do you trust what you see here?”, “Keep questioning…”] currentDialogueIndex = 1 questsToGive = [2, 7, 9]
DialogueManager	Manages dialogue appearance and progression	DisplayDialogue(Character character)	dialogueBoxActive = true

		bool dialogueBoxActive	
Player	Stores information related to player health, xp, and stats.	Custom Class **Player** Sprite spriteMask String name; int health; int xp; float movespeed; Move() PickUpItem (on collision w/ item, space) AttackEnemy (on collision w/ enemy, space) Speak(on collision w/ NPC, space)	spriteMask = <player pixel art> name = “Cobalt” health = 100 xp = 2000 moveSpeed = 14
CalculusProblem	Stores information related to a singular calculus problem	Custom Class **CalculusProblem** String problem String solution int xpForSolving CheckAnswer(String playerAnswer)	Problem = “Find the derivative of $6x^2$” Solution = “12x” xpForSolving = 10
TLOCGameData.json	Stotes game information in the InventoryManager, QuestManager, JournalManager, and Player to file for persistence.	JSON file	N/A
DefaultTLOCGameData.json	Stores default values of TLOCGameData.json	JSON file	N/A

Algorithms:

InventoryManager:

```
AddToInventory(Item item)
    String itemName = item.GetItemName()
    for index in inventory:
        if inventory[index].GetItemName == itemName:
            Inventory[index].quantity += 1
            return
    //item not in inventory before so should be added
    inventory.append(item)

RemoveFromInventory(String itemName)
    try:
        for index in inventory:
            if inventory[index].GetItemName() == itemName:
                inventory[index].quantity = 0
                inventory.pop(index)
    catch:
        //wasn't in inventory, graceful error
        Debug.Log("Item wasn't in inventory to begin.")
```

QuestManager

```
StartQuest(int questIndex)
    activeQuests.append(quests[questIndex])
    journalManager.UpdateJournal()

EndQuest(int questIndex)
    //will be called when talking to NPC and check confirms all criteria completed
    activeQuests.remove(quests[int questIndex])
    completedQuests.append(quests[questIndex])
    journalManager.UpdateJournal()

UpdateQuestCriteria(int questIndex, String questCriteria)
    quests[questIndex].questCriteria[questCriteria] = true
    journalManager.UpdateJournal(questIndex)
```

JournalManager

```
UpdateJournal(int questIndex)
    currentEntryIndex = questIndex.journalEntries[currentEntryIndex + 1)
```

DialogueManager

```
On KeyPress Space:
DisplayDialogue(Character character)
    if not dialogueBoxActive:
        dialogueBoxActive = true
    character.currentDialogueIndex += 1
```

```
if character.currentDialogueIndex == character.dialog.size:
        currentDialogueIndex = 0
        dialogueBoxActive = false
```

Test Plan:

Step Description	Input	Expected Outcome
User launches game	None	Main menu screen appears with options to enter “Options” or play game. Background music plays.
User clicks “Options” button	Button click	Options page opens with edit-able settings.
User adjusts volume	Slider clicked and slid.	Volume changes to match slider volume.
User enables/disables streak	Radio button selection	On the main menu, a streak UI either appears or disappears.
User resets game data	Button click and affirmative answer to prompt	TLOCGameData.json is reset to match DefaultTLOCGameData.json <see DataTable>
User exits Options	Button click	Volume, streak options saved to TLOCGameData.json. Options page closes and Main Menu page reappears.
User starts game	Button click	Game begins; Main Menu page disappears and game world loads in. Player starts in the main hall.
Player moves	Arrow keys or WASD	Player sprite moves in correct direction; sprite and animations adjust accordingly.
Player picks up item	Collision with item and SPACE pressed	Item disappears from game world (w/ effect ideally). Item added to inventory.
Player opens inventory	I key pressed	Inventory window opens overlay. Small boxes contain a tiny sprite of the item. Scroll overflow. Game world pauses in background.
Player scrolls over inventory item	Mouse over	Textbook detailing item quantity and description appears beside mouse. Disappears when mouse off.
Player closes Inventory	I key pressed	Inventory window overlay closes. Game resumes.
Player speaks to NPC	Collision with NPC boundary and SPACE pressed	Game pauses. Dialogue box appears at bottom of screen with correct NPC dialogue. Continuous pressing of SPACE steps through dialogue. When dialogue finishes box disappears and game resumes.

Player is offered Quest	NPC spoken to has a quest to give	NPC displays Quest offer dialogue and a yes/no prompt appears for the player to accept quest.
Player accepts quest	“Yes” button clicked on prompt	Quest object moved to activeQuest array in QuestManager. NPC finishes explanatory dialogue. Journal updated.
Player declines quest	“No” button clicked on prompt	Dialogue/prompt window closed. Game resumes.
Player opens journal	J key pressed	Journal overlay opens. Game pauses. Sidebar with quest titles. Completed quests at bottom of sidebar.
Player opens specific entry	Button click	Right half of Journal overlay populates to display relevant quest info.
Player closes journal	J key pressed	Journal overlay disappears. Game resumes.
Player completes quest	Quest criteria array all values set to true. NPC returned to.	Quest index added to completedQuest array and out of activeQuestsArray. Journal updated and entry index added to completedEntries array. Player XP increased OR item added to inventory.
Player begins fight with monster/creature	Collision with Enemy awareness box collider.	Game pauses. Combat overlay opens. One half the overlay has pixel art of the attacking Enemy. The other half displays a calculus problem. A timer counting down from timeForPlayerToSolveAttack. At the bottom of the screen is a text box for the Player to enter solution.
Problem generated	Attack sequence begins *repeated throughout encounter*	New random problem and solution generated by program.
Player enters solution to Enemy problem.	Text typed in textbox and ENTER pressed	Textbox cleared. Solution checked against problem answer.
Player entered correct solution to defense	Submitted solution was correct	Player dodge animation played. Player avoids taking damage.
Player entered incorrect solution to defense	Submitted solution was not correct	Player hurt animation played. Health bar decreases.
Player solves a problem to attack Enemy	Defensive combat sequence finished	New problem populates half of overlay. Timer now counts up from 0.
Player enters correct solution to attack	Submitted solution was correct	Player attack animation played. Enemy hurt animation played. Enemy received damage - seconds taken to answer problem.

Player enters incorrect solution to attack	Submitted solution was incorrect	Player attack animation played. Enemy dodge animation played.
Player dies	Player health reaches 0	Most recent TLOCGameData.json dated hot loaded.
Player wins encounter.	Enemy health reaches 0	Enemy defeat animation plays. Combat overlay windows closes. Player receives XP OR item added to inventory.
Player unlocks story bit	XP threshold reached OR important quest completed	A new dialogue opportunity revealing more of the story appears in the main hall.
Player unlocks main door	All pebbles of calculus are in inventory and SPACE pressed upon collision with door box collider.	Door opens, Player can access final game area. Final Quest prompted to begin.
Player completes game	Final Quest completed.	Game saved. Credits played. Player informed can keep playing side quests. Player reloaded to main hall with most recent save. Post-game area unlocked.
Player opens Pause page	P key pressed.	Pause page overlay opens. Game pauses in background.
Player saves game	Button click.	TLOCGameData.json updated with proper values.
Player quits to main menu	Button click	Game saves. Game world disappears. Main Menu page appears.
Player quits to desktop	Button click	Game saves. Program closes.

Design Complexity:

1. **Arrays:** arrays are used in managers to keep track of player inventory, active/completed quests, as well as in methods to step through dialogue lines and check off quest criteria.
2. **Custom Classes:** custom classes are used for Quests, Items, Enemies, Characters, the Player, and JournalEntries.
3. **File Input:** save data is kept in a .json file which is loaded when the game starts
4. **Loops:** C# scripts each have an Update() method that runs on every frame of the game. Loops are used for finding/removing/adding Items or Quests to their respective managers.
5. **Conditionals:** umm they're used a lot. They're kind of an essential part of programming. I'm not going to list all the different places I'm going to use conditionals. They will be used a lot.