## Character

* Characters
  + What is needs:
    - Empty gameObject (reset transform)
    - Sprite (different colored squares for now)
    - Animator
    - Script (tentatively named CharacterTemplate) with variables:
      * Character functions for taking damage and receiving buffs and debuffs
      * **The ScriptableObject CharacterData charaData which will contain:**
        + **spriteRenderer**

Character Sprite

* + - * + **Scriptable Object MoveList (carries all the moves. Equipped with asset menu)**

**Create New**

**Moveset**

**Elru’s moveset**

**Put all moves into this move list**

**Sounds easy enough?**

* + - * + **[Serialize Field] Statistics characterStats (Stats script to hold each character’s stats)**

Individual lists for these certain lists:

Elru’s stats

* + - * + **Character AnimationClips Array** (not now, but will be in there)
        + Character Images Array (array for all the images charas have when having a move chosen, item, things like that)

Most likely 2 images- ATK and ITEM, then what, of each, they’re better at

ITEM for Lola would be her using her tools well, since she’s good at that

ATK for Lola sees her using her weapon, since she’s good at that too

ATK for Elru would have him looking smug with bubbles of water surrounding him

ITEMS for Elru would be him looking wholly confused with potions in his hands since he’s good at nothing in items

ATK for Zeffre would be him singing with music notes because he’s good at support

ITEMS for Zeffre depict him a bit confused with potions since he doesn’t seem to be good at them

Have the images change if the characters stats have been acclimated to something they weren’t usually good at due to stat boosts?

* + - * + SpriteRenderer of the child for animations

private SpriteRenderer childSpriteRenderer;

// Get the child object directly from the parent

GameObject childObject = this.transform.GetChild(0).gameObject;

// Get the SpriteRenderer component of the child

childSpriteRenderer = childObject.GetComponent<SpriteRenderer>();

* + - * + **Bool characterDowned**
        + gameObject(?)

weaponSlot

itemSlot1

itemSlot2

* + - * + **Enum charaStatus** (keeping track of what status affliction you have. To be moved?)

FREEZE

STILL (Zeffre and Lola?)

POISON

Poison power depends on ability? Will have to mull DoT statuses so not yet.

BURN

NONE

* + - Remember to Prefab this to make other characters. Had better make the barest model with all these components

## Moves

* Individual moves
  + **Scriptable object, NOT Monobehavior**
  + **Create asset menu fileName=”New Move” menuName=“Moves/New Move”**
  + **Float (can convert to int when all is said and done)**
    - **movePower**
  + **Int** 
    - **mpCost**
    - **lvlGotten**
  + **String**
    - **moveName**
    - **moveTypeDesc**
      * **Could be a symbol, so may have to be a gameObject variable or Image var**
    - **movePowerDesc**
    - **mpCostDesc**
    - **moveExplanation**
  + **Enum**
    - **attackType[] (could make an array to make it phys/status or mag/status or pure status)** 
      * **PHYSICAL**
      * **MAGICAL**
    - **moveType[] (array for if the move is damaging, supplementary, or healing)**
      * **DAMAGING**
      * **SUPPLEMENTARY**
      * **HEALING**
    - **statusType[] (array for multiple statuses. Will always be towards enemies)**
      * **FREEZE**
      * **STILL (Zeffre and Lola?)**
      * **POISON**
        + **Poison power depends on ability? Will have to mull DoT statuses so not yet.**
      * **BURN**
      * **CURE**
      * **NONE**
    - **Boost[] (for buffs and debuffs)**
      * **DEFENSE**
      * **ATTACK**
      * **MAGIC**
      * **SPEED**
      * **SKILL**
      * **NONE**
    - **(tentative targetNumber)**
      * **ONE**
      * **ALL**
        + **If target is ALL, don’t pick the target, null will turn into all enemies or allies (that aren’t downed?)**
* Character Movesets
  + **Scriptable object, NOT Monobehavior**
  + **Create asset menu fileName=”New Moveset” menuName=“Moves/New Moveset”**
  + **List<Moves> characterMoveset**

## Stats

* **Stats List**
  + **Scriptable object, NOT Monobehavior**
  + **Create asset menu fileName=”New Character Stat” menuName=“Statistics/New Character Stat”**
  + **String**
    - **characterName**
      * **name**
  + **Character Ability? Think more on this** 
    - **String characterAbility**
      * **ability**
    - **abilityDesc**
      * **abilityDesc**
    - Then have a script that goes through abilities and says what they do
  + **All floats (can convert to int when all is said and done)**
    - **IMPORTANT: Base and Current for all of these (we need to keep an eye on the raising and/or lowering of these. Should we have stats with armor too? Maybe just, in the beg of the battle, make the stats current stats = base stats with stats given by armor)**
      * **HP (Hit Points. How many attacks the character can take before they are downed)**
        + **baseHP**
        + **currentHP**
      * **MP (Mana Points. How many special moves can be cast until they can no longer use them)**
        + **baseMP**
        + **currentMP**
      * **Atk (for physical offense. The higher it is, the better you can hit with physical attacks)**
        + **baseAtk**
        + **currentAtk**
      * **Def (physical defenses. The higher it is, the better you can resist physical attacks)**
        + **baseDef**
        + **currentDef**
      * **Mag (for magical offense. The higher it is, the better you can hit with magical attacks)**
        + **baseMag**
        + **currentMag**
      * **Res (magical defenses. The higher it is, the better you can resist magical attacks)**
        + **baseRes**
        + **currentRes**
      * **Skill (the odds that you can crit)**
        + **baseSki**
        + **currentSki**
      * **Efficiency (how well you can use offensive items, or potions)**
        + **baseEff**
        + **currentEff**
        + **mixes with offensive stats to make sure healers aren’t secretly monsters**
        + **For offensive tools, scales off attack or magic to differentiate attackers from healers**
      * **Spd (the higher it is, the faster someone can attack. Highest speed attacks first)**
        + **baseSpd**
        + **currentSpd**
        + **Things to note:**

**High efficiency means high potion usage as well as high tool use**

**Great healers will have a high magic stat, to offset, we give them low power or no magic attacks**

* + **Enum**
    - **Immunity[] (what the character can resist)**
      * **FREEZE**
      * **STILL (Zeffre and Lola?)**
      * **POISON**
        + **Poison power depends on ability? Will have to mull DoT statuses so not yet.**
      * **BURN**
      * **ALL**

## Items

* Scriptable Object ItemObjects
  + **Enum**
    - **itemTypes**
      * **RESTORATION**
      * **PERMSTATBOOST**
      * **EQUIPITEM**
      * **WEAPON**
      * **TOOL**
      * **DEFAULT**
      * **JOKE**
  + **String**
    - **itemName**
    - **itemPlural**
    - **[text area] desc**
  + **Int**
    - **itemPrice**
  + statusType[] (array for multiple statuses. Will always be towards enemies)
    - FREEZE
    - STILL (Zeffre and Lola?)
    - POISON
      * Poison power depends on ability? Will have to mull DoT statuses so not yet.
    - BURN
    - CURE
    - NONE
  + Bool
    - (tentative. Will not be used for a while) ableToSell
* These scripts need to inherit ItemObject, NOT Monobehavior
  + **Restoration Object (in battle inventory in battle. restores hp, mp, or both)**
    - **Float (can convert to int when all is said and done)**
      * **mpRestoration**
      * **hpRestoration**
  + **Perm Stat Boost Object (in inventory. permanently raises one or more character stats)**
    - **Float (can convert to int when all is said and done)**
      * **hpBoost**
      * **mpBoost**
      * **atkBoost**
      * **defBoost**
      * **magBoost**
      * **resBoost**
      * **skillBoost**
      * **effBoost**
      * **spdBoost**
  + **Equip Object (can fit into slots and raise character stats for as long as it is equipped (can be fused with weapon object?))**
    - **Float (can convert to int when all is said and done)**
      * **hpBoost**
      * **mpBoost**
      * **defBoost**
      * **resBoost**
      * **skillBoost**
      * **effBoost**
      * **spdBoost**
  + **Weapon Object (tentative: can raise attack or defense as long as it is equipped)**
    - **Float (can convert to int when all is said and done)**
      * **atkBoost**
      * **magBoost**
        + Slot will be made exclusively for weapon, the rest will be equips
  + **Tool Object (in inventory in battle. Used to damage enemies and give status effects(?))**
    - **Float (can convert to int when all is said and done)**
      * **atkPwr**
      * **magPwr**
  + **Default Object (normal items. Probably have bits of lore to them)**
    - **No changes**
  + **Joke (to be added later, a weapon, when equipped to a character whose name matches characterName, increases their best stat by far, more than exclusive weapons)**
    - **string**
      * **trueDesc (shows when equipped to the right character)**
    - **Ability (tentative)**
  + **Exclusive (to be added later, a weapon, when equipped to a character whose name matches characterName, increases their best stat by far, more than exclusive weapons)**

## Inventory

* Holds all character’s items
  + Create asset menu fileName=”New Inven” menuName=“Inventory/New Inventory”
  + Scriptable Object
    - List <Items>
* Add and remove inven items
* Transfer inven items from shop to player inven
* Ability to sell items
* Add or remove inven items
* Have store appear after battle set

## Main UI

* General setup of the UI
* Make Canvas called UI
* Empty Game Object (please reset transform and expand to fit a whole UI)
  + **Image for UI HUD**
    - **Main HUD empty Game Object (on the left, give space to the buttons)**
    - **NEW IDEA for the main HUD**
      * **Player Transform**
        + **This is in the area**
      * **Player section empty Game Object (ONLY Once) This will be the panel.**
        + **Player Name**

**Name TMP**

* + - * + **Player Level**

**Level TMP**

* + - * **HP Empty GameObject**
        + **White Background**

**White image**

* + - * + **Player HP Bar**

**Green slider smaller than white image**

* + - * + **Player HP Pain bar**

**(tentative) Red slider smaller than white image**

* + - * + **Player HP Count**

**( looks like 0/0) HP number TMP**

* + - * **MP Empty GameObject**
        + **White Background**

**White image**

* + - * + **Player MP Bar**

**Blue slider smaller than white image**

* + - * + **Player MP depletion bar**

**(tentative) yellow slider smaller than white image**

* + - * + **Player MP Count**

**( looks like 0/0) MP number TMP**

**Turn this panel into a prefab**

* + - **Button HUD empty Game Object (**[**Button menu navigation**](https://www.youtube.com/watch?v=8mFWJwxV3ps)**)**
      * **Attack Button (top button: Attack, bottom button: Supp)**
        + **ATKButton**
        + **ATKTMP**
      * **Supplementary Button (top button: Attack, bottom button: Items)**
        + **SUPPButton**
        + **SUPPTMP**
      * **Items Button (top button: Supp, bottom button: Stats)**
        + **ITEMButton**
        + **ITEMTMP**
      * **Stats Button (top button: Items, bottom button: Forfeit)**
        + **STATButton**
        + **STATTMP**
      * **Forfeit (top button: Stats, bottom button: Attack)**
        + **QUITButton**
        + **QUITTMP**
      * **REMEMBER TO KEEP THE BUTTONS HIGHLIGHTED, EVEN IF ONLY A LITTLE**

## UI Manager Script

* Set up UI here
  + (This will grab all objects and buttons found on the UI) Might want to place the Characters first
  + Holds all that a UI needs to, from the boxes to the actions
    - gameObjects
      * **For HUD**
        + **attackAndInvenBox**
        + **subUIBox**
        + **statsBox**
        + **quitBox**
        + **allyAndEnemyBox (to be used in a different script)**
      * For UI
        + **charaImage;**
    - **Sliders**
      * **playerHPSlider[]**
      * **playerMPSlider[]**
      * **allyHPSlider[]**
      * **enemyHPSlider[]**
    - TMPs
      * For UI
        + **playerNames[]**
        + **playerLvls[]**
        + **playerHP[]**
        + **playerMP[]**
        + To grab the var that we can change the text name with (.txt = “this”)

atkTMP

suppTMP

itemTMP

statsTMP

quitTMP

* + - * **For Stats**
        + **abilityText;**
        + **descAbilityText;**
        + **nameText;**
        + **levelText;**
        + **hPText;**
        + **mPText;**
        + **atkText;**
        + **defText;**
        + **magText;**
        + **resText;**
        + **effText;**
        + **skiText;**
        + **speText;**
        + **pointsToLevelText;**
      * **Image**
        + **characterImage**
  + **Buttons are in the button controller**
  + **Functions that go here**
    - **Set Stats Text for stats function**
      * **Parameters**
        + **TMP\_Text text**
        + **Int current stat**
        + **Int base stat**
        + **String stat text**
      * **Local**
        + **Int statDifference**
        + **String sign**
      * **Steps**
        + **statDifference has us find the difference between the current and base stat and that’s it (finding what was lowered or raised in the base)**
        + **Sign: if the stat difference is a negative number, - will appear, if not, + will appear, turning the number that color too**

**string sign = (statsDifference >= 0 ) ? $"<color=#6EFFFF>+{statsDifference}</color>" : $"<color=#FF1100>{statsDifference}</color>";**

* + - * + **Set the textValue.text to the statText, the base value, and the sign followed by the stat difference**

**statText: baseValue(sign statDIfference)**

**textElement.text = $"{statText}: {baseValue}({sign})";**

* + - **Show Stats function**
      * **Parameters**
        + **We need the CharacterStatistics charaStats**
      * **Tbh set everything**
        + **charaImage.GetComponent<Image>().sprite = characterStatistics.Image;**
        + **Set the text for the ability, desc, and chara name**

**abilityText.text = characterStatistics.Ability;**

**descAbilityText.text = characterStatistics.AbilityDesc;**

**nameText.text = characterStatistics.CharacterName;**

* + - * + **Set the level as a string**

**levelText.text = "Level: " + characterStatistics.Level.ToString();**

* + - * + **Set hp and mp as 0/0 for current and base health**

**hPText.text = $"Health: {characterStatistics.CurrentHP}/{characterStatistics.MaxHP}";**

**mPText.text = $"Mana: {characterStatistics.CurrentMP}/{characterStatistics.MaxMP}";**

* + - * + **Use the function SetStatText for the main stats. Include the parameters for the stat’s:**

**TMP\_Text**

**Current stat**

**Base stat**

**The string name you want to set the stat to**

* + - **Set up HUD**
    - **Player HUD**
      * **We’ll need to instantiate the panels in order to match the number of players of the current party. So, new plan.**
        + **Global int characterIndex = 0**
        + **Button Generator script**
        + **We’ll need a void InstantiateCharacterPanels**

**We need GameObject panelPrefab so we can hold the single prefab we made of the character panel.**

**Transform panelTransform to hold where our prefabs are going to be**

* + - * + **Parameters will take a list that holds CharacterData “party”, as we’ll put the current party in this without exception.**

**Foreach loop to grab each member in the “party”**

**GameObject “panel” will Instantiate panelPrefab in panelTransform**

**Grab the children of the panel to manipulate them (panel.transform.GetChild(0).GetComponent<GameObjectt>()).**

**This is where the child lies in the parent. 0 is the first element, 1 is the second, and so on. So these NEED to be in order**

**Name**

**TMP\_Text. Name = panel.transform.GetChild(0).GetComponent<TMP\_Text>()).**

**Level**

**TMP\_Text. Level= panel.transform.GetChild(1).GetComponent<TMP\_Text>()).**

**Healthnum**

**TMP\_Text. hpNum= panel.transform.GetChild(2).GetComponent<TMP\_Text>()).**

**Mpnum**

**TMP\_Text. Name = panel.transform.GetChild(3).GetComponent<TMP\_Text>()).**

**White bar HP**

**GameObject whiteBarHp = panel.transform.GetChild(4).gameObject**

**Hp slider**

**Slider hpSlider =panel.transform.GetChild(0).GetComponent<TMP\_Text>().**

**White bar MP**

**GameObject whiteBarMp = panel.transform.GetChild(5).gameObject**

**Mp slider**

**Slider mpSlider=whiteBarMp .transform.GetChild(0).GetComponent<Slider>().**

**There’s no space anymore, so we’ll do this here. We’ve grabbed each component, now we need to make their text match what we want**

**Make the name.text match the name in the “member’s” name in its stats list**

**Level matches the member’s level to string**

**Hp matches the member’s current hp / max hp**

**Mp matches the member’s current mp / max mp**

**Slider’s max health and magic equals the max health of the hp and mp**

**Same with the member’s current hp and mp with the slider’s regular value (value)**

* + - * **Make sure to space the panels out (check move generator?)**
    - **Allies HUD (this will be about the same thing and a separate function)**
      * **Still grab the character data list as a parameter**
      * **Panel prefab for allies this time. A panel transform as well.**
        + **For each “member” in “party” still**

**Instantiate the panel at the transform**

**Grab the children of the panel. They should be**

**Button (child 0)**

**Name (child 0 of the button)**

**White panel hp (child 1)**

**Slider hp (child 0 of white panel child)**

**White panel mp (child 2)**

**Slider mp (child 0 of white panel child)**

**Make the text and sliders match what we want**

**Make sure the button has a value so that, when we press it, it’ll be whichever party member matches the current value**

**Will have to AddListener (buttonGenerator.OnPlayerClick(characterIndex))**

**characterIndex++**

* + - * **Panel prefab for enemies. Just a slight difference**
        + **Near the exact same as the allies panel. Tbh we could take the transform of the allies panel to do this. What we want is**

**Button**

**Name**

**White bar hp**

**Hp slider**

**Do the same thing with player click but with OnEnemyClick(characterIndex)**

* + - * Should there be an onClick for this button?
      * Most likely.

## Sub UI

* The windows that come up when pressed
  + SubUI Box (No need for an empty game object for this, i don’t think. Also, used only if I figure out a way to have all actions used one by one)
    - **For Moves and Items**
      * **MoveAndItemHolder empty Game Object**
        + **Move and Item button**
        + **Shape it to how you want it to be**
    - **(Tbh may need its own box for the sheer size of what I’m not thinking it’s going to be.)**
    - **For Stats**
      * **Stats Empty Game Object**
        + **Image**

**Player Image**

* + - * + **TMP**

**Player Stats Text**

**All INDIVIDUAL stats TMP (like Attack: 7(+0))**

**Please do these for all stats**

**Points to Level TMP**

* + - * + **Button**

**Next Button**

**Next TMP**

**Back Button**

**“Return” TMP**

* + Quit Box
    - Quit Empty Game Object
      * Are you sure you want to leave? TMP
      * Yes Button
        + Yes TMP
      * No Button
        + No TMP
  + **Allies Box (for when you want to heal or power up allies)**
    - **Ally Image “panel”**
      * **Ally Empty Game Object**
      * **Ally1-4 hp bar**
        + **1 white bar**
        + **One green bar**
  + Could make the ally and enemy box the same , just turn the gameObject off whenever needed, if not:
    - **Enemies Box (for when you want to attack or debuff up enemies. May not be needed because above)**
      * **Enemy Image “panel”**
        + **Enemy Empty Game Object**
        + **Enemy1-4 hp bar**

**1 white bar**

**One green bar**

## Character Placement

* **This is to place the characters and enemies. This most likely goes before UI, so we can calc the space needed to place the characters**
* **Make ONE single gameobject Character.**
  + **Has a script called CharacterTemplate**
  + **Make into a prefab**
* **Player Stations Empty Game Object (as big as needed. Please reset the transform)**
  + **Player1-4 Stand**
    - **Empty Game Object (reset transform)**
      * **Player sprite, move to designated place you’d want the player being in that doesn’t ho the others.**
        + PLEASE talk to Elle about character sizing when the time comes
      * **Prefab the player sprites, then delete them from the game view. All we need are the stations. These are players 1-4.**
        + **Repeat from here with the enemies**

## Button Generator

* Generates buttons for the attacks, items, and other buttons we might need
* Sub UI Generation Script
* Have a function that, when the box is set active, the moves and inventory appear. When the box is no longer active, the moves and inven are gone. This way, I can only have one box instead of like 4 for attack, supp, items, and stats.
  + if(button is pressed && boxActiveOn(bool for setActive false)
    - Shut off all other sub UIs but that one
      * Can be used with CancelInvoke(“theMethodName”)
        + Or just make 2 windows for ATK and ITEMS and justify them being different. Seems like less of a hassle, tbh.
* Generate Attacks (including supplementary attacks as well)
  + What we need
    - GameObject
      * button prefab (needed for literally both moves and items)
    - Transform
      * atkButtonContainer;
      * itemButtonContainer;
    - Float
      * buttonSpacing;
    - playerInven;
    - bool
      * attackButtonPressed = false;
    - Moveset
      * selectedMove;
    - ItemObject
      * selectedItem;
    - List<Moveset> movesAlreadyAdded = new List<Moveset>();
  + Make a method that takes the character unit’s characterData’s moveLlist (unit[i].moveSet). This is needed to access their move list.
    - public void GenerateATKButtons(Moveset characterSet)
  + Needed locally
    - float currentPosY = 0f;
  + Clear the list movesAlreadyAdded
    - movesAlreadyAdded.Clear();
  + To make room for the next character, destroy the other buttons in the atkContainer
    - foreach (Transform button in atkButtonContainer)
    - {
    - Destroy(button.gameObject);
    - }
  + Just in case, say if the moveset isn’t null
    - if (characterStatistics != null)
  + Cycle through the moveset’s count
    - for (int i = 0; i < characterStatistics.moveBaseClassList.Count; i++)
  + Hold each move. We’re going to filter it
    - MoveBaseClass move = characterStatistics.moveBaseClassList[i];
  + If the character has the move’s level, and the list movesAlreadyAdded doesn’t already that move.
    - if (characterMoveset.Level >= move.LevelAqcuired && !movesAlreadyAdded.Contains(move) && move.MoveType == MoveType.DAMAGING)
  + Instantiate the button prefab in the atkContainer
    - GameObject buttonGO = Instantiate(moveButtonPrefab, atkButtonContainer);
  + Move where the button is GOING to be so we can give the next button space
    - buttonGO.GetComponent<RectTransform>().anchoredPosition = new Vector2(0f, -currentPosY);
    - currentPosY += buttonSpacing + buttonGO.GetComponent<RectTransform>().sizeDelta.y;
  + Not done yet, name the button the name of the move.
    - buttonGO.GetComponentInChildren<TMP\_Text>().text = move.AttackName;
  + Add the move to movesAlreadyAdded
    - movesAlreadyAdded.Add(move);
  + Grab the button through code, we need to add a method
    - Button buttonComponent = buttonGO.GetComponent<Button>();
    - buttonComponent.onClick.AddListener(() => OnAttackButton(move,i));
* Generate Items (only consumables and throwing items)
  + Mostly the same things, but we won’t need parameters for this method
  + Make sure we’re using the count of the right inventory
    - for (int i = 0; i < playerInven.Container.Count; i++)
  + Grab the specific inventory we need
    - InventorySlot slot = playerInven.Container[i];
  + Now hold each item, we’re going to filter through
    - ItemObject item = slot.item;
  + If the item is a consumable or a tool
    - if (item.Type == ItemType.Health || item.Type == ItemType.DamagingTool)
  + Make the button again and spacing
  + Amount was already figured out in amount inven, so we’re just saying how much we have here
    - buttonGo.GetComponentInChildren<TMP\_Text>().text = item.InvenItemName + " x" + slot.amount;
  + Making a new method
    - buttonComponent.onClick.AddListener(() => OnItemButton(item, i));
* OnAttackButton
  + Need Move selectedMove parameter
  + If by ANY chance you can access this window when you’re not on the player’s turn, just…don’t.
    - if (battleSystem.state != BattleState.PLAYERTURN)
    - {
    - return;
    - }
  + When this button is pressed, save the move, say the button has been pressed, and turn off the attack window
    - selectedMove = move;
    - attackButtonPressed = true;
    - attackandSupplementary.TurnOffButton(); (will be changed, but it uses a turn off method)
* OnItemButton
  + Same as above but with items
* clears the flag that signals when the player has made their move
  + public void ClearAttackButtons()
  + {
  + attackButtonPressed = false;
  + }
* check if the attack button's been pressed
  + public bool HasPressedAttackButton()
  + {
  + return attackButtonPressed;
  + }

## **Saving Attacks**

* **Saving player choice of attack**
* **Saving the unit, item or move used, and the target to use in the Battle Phase**
* **Plan (cause I need to understand what and why I’m doing this):**
  + **We’re going to make a class that has the unit attacking, the move or item used, and the targeted unit.**
    - **That class will have two constructors that save the actions of the character. These will be named the exact same thing as they will be overrides of each other. The only difference will be the second parameter being either a move or item being used.**
    - **Outside of the class (so above), that will store a list of literally that class. We’ll need that for the battle system.**
    - **We’ll be making 2 methods below that that’ll add a new SaveAction to the list above. It’ll have the parameters needed to save the unit, move, item, and target, saving them to the list.**
      * **We might also need to debug.log what the chara is doing and how many characters have used moves as to not go over**
  + **Needed**
    - **Class SavedActions**
      * **CharacterData** 
        + **character**
        + **target**
      * **ItemObject**
        + **item**
      * **Move**
        + **move**
      * **public SaveActions constructor #1**
        + **Parameters**

**CharacterData \_character**

**CharacterData \_target**

**Move \_move**

* + - * **public SaveActions constructor #2**
        + **Parameters**

**CharacterData \_character**

**CharacterData \_target**

**ItemObject\_item**

* + - **List<SavedActions> actionContainer = new List<SavedActions>**
    - **Functions**
      * **SaveCharaActions Method**
        + **Parameters**

**CharacterData \_character**

**CharacterData \_target**

**Move \_move**

* + - * + **Save the actions while debugging what’s being done and saying the list count to make sure we have the right amount of moves**

**actionContainer.Add(new SavedActions(\_character, \_target, \_move))**

**debug.Log(“Number of characters that have used moves: ” + actionContainer.Count)**

**debug.Log($ “{*character.charaStats.name}* is going to use the move {*move.name*} on {*target.charaStats.name*}.“ )**

* + - * **SaveItems**
        + **Parameters**

**CharacterData \_character**

**CharacterData \_target**

**Move \_item**

* + - * + **Save the actions while debugging what’s being done and saying the list count to make sure we have the right amount of moves**

**actionContainer.Add(new SavedActions(\_character, \_target, \_item))**

**debug.Log(“Number of characters that have used moves: ” + actionContainer.Count)**

**debug.Log($ “{*character.charaStats.name}* is going to use the move {*item.name*} on {*target.charaStats.name*}.“ )**

## Battle System

1. **Player Setup**
   * **Instantiate all characters, be it enemy and ally**
     + **Scripts needed**
       - **Party List**
       - **Damage Cals**
       - **Saving Attacks**
       - **UI Manager**
       - **Move Generator**
       - **Button Controller**
   * **Will need to instantiate certain enemies in their waves**
     + **“Randomly” generated enemies? I mean have them in a wave Scriptable Object that holds enemies. In that way, I’m thinking you can choose from 1 to 4(?) enemies to instantiate, then instantiate them based on the enemies**
     + **Have a level system that has lvl 1 stats, then, for each lvl up, stats go up (don’t have to manually boost stats for enemies that way)**
   * **Instantiate characters:**
     + **Need Party List script**
     + **Need gameObject characterTemplate as a global var**
       - **We still have our CharacterTemplate script**
     + **Put the characterData of the current party in List<gameObject> currentParty**
     + **When instantiating, instantiate the character gameObject multiple times. Instantiate them at their stations. Each player will have their characterData assigned to them**
     + **(Also in book. Check book as well)**
       - **gameObject characterPrefab**
       - **Transform[] playerStations**
       - **Int i = 0, i < currentParty.Count, i++**
         * **Transform spawnStation = playerStations[i];**
         * **GameObject charaObject = Instantiate(charaPrefab, spawnStation)**
         * **CharacterTemplate unit= charaObject.GetComponent<CharacterTemplate>()**
         * **unit.characterData=partyList.currentParty[i] (remember, this is character data as well)**
         * **// Set the character sprite**
         * **unit.spriteRenderer.sprite = unit.characterData.sprite;**
       - **When that’s done, start the coroutine EnemySetup**
2. **Enemy Setup**
   * **If battle is won, then have the enemies set up again, but use the next battle’s characters. Increase the battle count by 1**
     + **When instantiating, instantiate the character gameObject multiple times. Instantiate them at their stations. Each enemy will have their characterData assigned to them**
   * **Need a battle counter? Most likely**
     + **Int**
       - **currentBattle = 0 (needed?)**
       - **currentSeed= 0**
       - **currentCheckpoint= 0**
     + **Transform[] enemyStations**
     + **Go into the PartyList Script and grab the checkpoint, seed, and group**
       - **It would look like going into checkpoint (the List variable’s name) 1, seed 2, battle 3**
         * **That’s the battle we’re grabbing- battle 3. So we need battle 3’s Count. Can’t make this long thing its own variable, so we’ll have to make do.**
       - **So like if(int i = 0; i < partyList.checkpoint[currentCheckpoint].seedGroup[currentSeed].group.Count; i++)**
       - **Transform spawnStation = enemyStations[i];**
       - **GameObject charaObject = Instantiate(charaPrefab, spawnStation)**
       - **CharacterTemplate eUnit= charaObject.GetComponent<CharacterTemplate>()**
       - **eUnit.characterData=checkpoint[currentCheckpoint].seedGroup[currentSeed].group[i] (remember, this is character data as well)**
       - **// Set the character sprite**
       - **eUnit.spriteRenderer.sprite = eUnit.characterData.sprite;**
3. Player Turn
   * Player 1 start
     + Button pressed on the UI (ATK, ITEMS, those 2 will continue the sequence, not STATS)
   * A for loop that has i = our currentCycleIndex (used for the undo button), and i being less than our currentParty.Count
   * For attacks
     + Generate attacks for that player when the attack button is pressed (done. The OnATKButton, when pressed, should activate, letting the code know if you want to attack or use an item)
       - Choose an attack.
         * If the move has more than one buff, it’s for an ally.

Open the ally panel that’s in the button controller

The player will click the ally, so yield wait until selectedCharacter isn’t null

Save player choice of attack in saving attacks script (save the move, character who used the move, and character who was the target)

SaveCharaActions Method

OnTurnOff from the button controller to turn off everything but the hud to get ready for the other moves

* + - * + If the move power if more than 1, or the move has more than one debuff, the move is for an enemy.

Open the enemy panel that’s in the button controller

The player will click the enemy, so wait until selectedCharacter isn’t null

Save player choice of attack in saving attacks script (save the move, character who used the move, and character who was the target)

SaveCharaActions Method

OnTurnOff from the button controller to turn off everything but the hud to get ready for the other moves

* + For items
    - Generate items for that player when the item button is pressed (done. The OnITEMButton, when pressed, should activate, letting the code know if you want to attack or use an item)
      * Essentially the same thing, but the if parameters are different
        + If the item’s item type is of health
        + If the item’s item type is of damage

Save results

* + At this point, close the all the screens, turn the selected chara, selected enemy, selected move, selected item null, moveGenerator.ClearAttackButtons(), and turn off the panels. Increment the current cycle index.
    - This gets us ready for the next characters
  + After that, get out of the loop. Yield wait a few seconds and debug.log “Decisions made”. Change the current cycle index to 0 and change the state to Enemy Turn.

1. Enemy Turn
   * AI for later. What needs to be done is:
     + Get the characters who haven’t been downed
     + Generate a random number from 1 to the number of characters
       - w/e number that is is the character[generatedNumber]
     + Generate a random number from 1 to the number of moves the enemy is set up with
       - w/e number that is is the move[generatedNumber]
     + Save that result
   * Go to battle calc
2. Battle Calculation
   * Gather all saved results and sort them by player speed- fastest goes first
   * Calculate damage in another method
     + This is the same script that will print the results and have characters take damage
   * Reset things(most likely in a method) so character 1 starts normally
     + selectedMove is null
     + selectedTarget is null
     + playerCount is 0
       - IMPORTANT: Need some kind of debug window to show that these are acting how they should
3. Win
   * YOU WON! Go to next wave or say “thank you for playing the demo”
   * If Checkpoint is reached, go to battle prep and shop(?). These need to be created.
   * Adds every battle one up to battle 10. Then reaches battle prep
   * Int
     + battleCount (NOT in this method, but global) = 0
     + Since this is win battle, battleCount++
     + If battleCount > 9
       - (battleCount = 0?)
       - “Checkpoint Reached!”
       - Go to battle prep window
4. Loss
   * Player lost. Go back to the latest checkpoint. Need to make a checkpoint system.
     + Checkpoint probably needs to start with battle prep menu
     + See how you would go back
       - Save level progress?
         * Save the level, stats(?), level number, and team?
       - battleCount = 0
       - But don’t checkpoint++
       - Setup Enemy Battle
5. Checkpoint
   * checkpointCount ++
   * Do all the battle prep here?
   * When battle prep is over, have a button that says “To next battle”
     + Are you sure?
     + Go to Setup Battle
     + battleCount = 0

* Damage calced in another method
* Other functions are needed in this script

## Button Controller

* Might be where all the buttons are, tbh
  + For UI (might want to have them labeled that way)
    - Scripts
      * ButtonGenerator
      * PartyManager
      * BattleSystem
      * UIManager
    - Int
      * currentPlayer
      * currentStatsIndex
    - **Buttons**
      * **ATK**
      * **SUPP**
      * **ITEM**
      * **STATS**
      * **FORFEIT**
    - Bool
      * wantsToAttack
      * wantsToItem
    - GameObject
      * Blocker (to block player from pressing anything else while attack button is up)
      * allyContainer
      * enemyContainer
    - Functions
      * Activate all the buttons when needed
        + OnATKButton

Pulling up the buttons for attacking

Have the current player here match the current player in the player’s turn

currentPlayer = bSystem.currentPlayer

Turn on blocker

Blocker set active false

Turn on attack box and turn off everything else

Set attackBox active

Item box set active false

Stats box set active false

Generate attack buttons method in “Battle Generator” script

generator.GenerateATKButtons.(pManager.currentParty[currentPlayer].moveset)

* + - * + OnITEMButton

Pulling up the buttons for items and tools

Turn on blocker

Blocker set active false

Turn on attack box and turn off everything else

Set itemBox active

Attack box set active false

Stats box set active false

Generate item buttons method in “Battle Generator” script

generator.GenerateITEMButtons.()

* + - * + OnSTATSButton

Pulling up the display for all the current character stats (ALL character stats will be for later, and not here)

Have the current player here match the current player in the player’s turn

currentPlayer = bSystem.currentPlayer

Have the current stats index here match the current player in the player’s turn

currentStatsIndex = bSystem.currentPlayer

Needed so we’ll have the right character index when we press the next button

Turn on blocker

Blocker set active false

Turn on stats box and turn off everything else

Set statsBox active

Item box set active false

set attackBox active false

Generate stats method in “UI Manager” script

generator.GenerateSTATS.(pManager.currentParty[currentPlayer].characterStats)

* + - * + OnNextSTATSButton

As stats are open, go to the next character’s stats.

Have the current player here match the current player in the player’s turn

currentPlayer = bSystem.currentPlayer

If the index isn’t equal to or higher then the current party, go ahead and increase the stats index

if(currentStatsIndex < pManager.currentParty.Count)

currentStatsIndex++

If the index is greater than or equal to the current party, the index needs to go back to 0

if(currentStatsIndex >= pManager.currentParty.Count)

currentStatsIndex= 0

Either way, show the battle stats

generator.GenerateSTATS.(pManager.currentParty[currentStatsIndex].characterStats)

* + - * + OnQUITButton

Giving you the option to quit the gauntlet (either the gauntlet or that battle which will take you all the way to the closest checkpoint)

* + - * + OnOpenAllyBox(Opening the box for the player to help the ally)

Make the ally box active

SetAllyHUD from the UI manager

* + - * + OnOpenEnemyBox()

Make the enemy box active

SetEnemyHUD from the UI manager

* + - * + OnTurnOff

Just turn everything off sans, like, the hud lol

* + - * + OnPlayerClick(int index)

//picking a target ally

Battle system’s selected player index will = pList at whatever index

Debug.log the character who was chosen. Name them by their name, and w/e the index was

* + - * + OnEnemyClick(int index)

//picking a target enemy

Same as above, but get the group we’re currently in

## Damage Calc

* Calcs damage of all characters by move or item type and power as well as character defenses, offenses, and such.
* Honestly, we’ll need the bare minimum for now
* Damage from moves method (CalculateMove(attacker, move, target)
  + Method to take the corresponding move’s magic from the character
    - Method will be in CharacterTemplate, taking the parameter move.mpCost
  + If the move’s move type is damaging- will need a switch statement. So grab the move’s movetype too
    - If the move is physical
      * Int damage will equal (int)Math.Ceiling(the equation we made)
      * Have the chara take damage
    - If the move is magic
      * Int damage will equal (int)Math.Ceiling(the equation we made)
      * Have the chara take damage
    - After the switch case, make sure that if damage is less than or = 0, damage will be 1.
  + If it is supplementary (status, buff, or debuff with no damage)
    - * Method to take it to buffs and debuffs (the attacker, the move, the target)
  + If it is healing
    - Int healing will (int)Math.Ceiling(the equation we made)
    - Have the chara get healed
  + If the move has a buff, debuff, or status, go to a method with the parameters (the attacker, the move, the target)
* Damage from items method (CalculateMove(attacker, item , target)
  + If the item is damaging
    - Make the item type more specifically a damaging tool
    - If tool is magical
      * Int damage will equal (int)Math.Ceiling(the equation we made)
      * Have the chara take damage
    - If tool is physical
      * Int damage will equal (int)Math.Ceiling(the equation we made)
      * Have the chara take damage
  + If the item is healing
    - Make the item type more specifically a healing object (HealthObject healingItem = (HealthObject)item;)
    - If the item has more than 0 hp restoration
      * Int damage will equal (int)Math.Ceiling(the equation we made)
      * Have the chara get healed
    - If the item has more than 0 mp restoration
      * Int damage will equal (int)Math.Ceiling(the equation we made)
      * Have the chara get healed
  + If the item gives a status, go to a method with the parameters (the attacker, the move, the target)
  + Take the item away from the inventory
* Method for buffs and debuffs
  + We’ll need strings
    - buffDisplay = “”
    - debuffDisplay = “”
    - statusDisplay = “”
  + If there are buffs
    - This will have a method for a switch statement in Character Template whose parameters will be the move’s buff type
    - Make a loop that with i being less than the move’s buff type.Length.
    - Have buffDisplay += the “move.buffType[i]”
    - We’ll be adding a comma. To prevent adding a comma after the last buff type. Add if the current index i is less than the the move.BuffTypes’s length - 1, then buffDisplay += “ , ”
    - Get out of the loop and finally buffDisplay += “ raised!”.
  + If there are debuffs
    - This will have a method for a switch statement in Character Template whose parameters will be the move’s buff type
    - Same as above
  + If there is status
    - This will have a method for a switch statement in Character Template whose parameters will be the move’s status type
    - Will think about this more. Most likely going to just be statusDisplay = “the move.Status or something,” POISONED! Or CURED, y’know?
  + If the move’s atk power or mag power is more than one
    - String message would = this person used this on this person! Did this much damage! + debuffDisplay + statusDisplay
  + If the move or item heals
    - String message would = this person used this on this person! Healed this much! +buffDisplay + statusDisplay

## Character Template Script

* **What methods and things are located in our CharacterTemplate**
  + **IEnumerator TakeDamage(int damage)**
    - **Decrements damage every certain amount of time**
    - **We’ll make float difference = the chara stats current hp - the damage**
    - **And while the chara’s current hp is more than difference:**
      * **Decrement chara’s current hp by 1**
      * **Access a method to update the chara’s health and magic in the HUD**
      * **Yield wait for seconds….like…. 0.05f**
      * **Still in the loop. Make sure that if the chara’s current hp is or is less than 0**
        + **The current hp = 0**
        + **The character isDefeated**

**Prolly have a method to defeat them or set their sprite to “Downed”**

* + - * + **Or else isDefeated is false**
  + **IEnumerator GiveHealth(int health)**
    - **Increments health every certain amount of time**
    - **We’ll make float difference = the chara stats current hp + the health**
    - **And while the chara’s current hp is less than difference:**
      * **Increment chara’s current hp by 1**
      * **Access a method to update the chara’s health and magic in the HUD**
      * **Yield wait for seconds….like…. 0.05f**
      * **Still in the loop. Make sure that if the chara’s current hp more than the max hp**
        + **The current hp = the max hp**
        + **break**
  + **IEnumerator TakeMana(int mp)**
    - **Decrements mpevery certain amount of time**
    - **We’ll make float difference = the chara stats current mp - the mp**
    - **And while the chara’s current mp is more than difference:**
      * **Decrement chara’s current mp by 1**
      * **Access a method to update the chara’s health and magic in the HUD**
      * **Yield wait for seconds….like…. 0.05f**
  + **IEnumerator Heal(HealthObject item, int result)**
    - **Float**
      * **hPAddition = the chara stats current hp - the result**
      * **mPAddition = the chara stats current mp + the result**
    - **If statement that sees if the health item’s hp or mp restore amount is more than 0**
      * **If the item’s hp restoration amount is more than 0**
        + **And while the chara’s current hp is more than the addition:**

**Increment chara’s current hp by 1**

**Access a method to update the chara’s health and magic in the HUD**

**Yield wait for seconds….like…. 0.05f**

**Still in the loop. Make sure that if the chara’s current hp is or is more than the max hp**

**The current hp = the max hp**

**Break out of this loop**

* + - * **Else if the item’s mp restoration amount is more than 0**
        + **And while the chara’s current hp is more than the addition:**

**Increment chara’s current mp by 1**

**Access a method to update the chara’s health and magic in the HUD**

**Yield wait for seconds….like…. 0.05f**

**Still in the loop. Make sure that if the chara’s current mp is or is more than the max mp**

**The current mp = the max mp**

**Break out of this loop**

* + **ApplyBuffAndDebuff(Boost[] boostTypes, intboostAmount)**
    - **Will need a switch statement**
      * **Foreach boost in boostTypes**
        + **switch(boostType)**

**case(Boost.ATTACK)**

**Equation then break**

**Do this for every boost available**

* + **RemoveBuffsAndDebuffs()**
    - **Needed so we can start a match with the base hp and mp**
      * **Make the chara stats of each stat = its base stat**
  + Update HP and MP
    - public void UpdateHealthAndMagic()
    - {
    - if(characterStats.UnitType == UnitType.PLAYERCHARACTER)
    - {
    - battleHUD.UpdatePlayerHPAndMP(this, characterStats.CurrentHP, characterStats.MaxHP, characterStats.CurrentMP, characterStats.MaxMP);
    - }
    - else
    - {
    - battleHUD.UpdateEnemyHPAndMP(this, characterStats.CurrentHP);
    - }
    - }

## Groups

* **Groups are made here, be it party, enemies, what have you**
  + **Create asset menu fileName=”New Group” menuName=“Group/New Group”**
  + **Group Script is a Scriptable Object that takes a list of GameObjects**
    - **List<GameObject> groups**
    - **Enum rewardsAvailable (for enemies only)**
      * **YES**
      * **NO**
    - **Int currency**
      * **Any value we want. If not an enemy group, leave the int at 0**
  + **Most likely gonna make 4 groups of these**
    - **currentParty**
    - **partyInventory**
    - **Roster**
    - **enemyGroups**

## Group Seed

* **Scriptable objects that group the enemy battles into seeds**
  + **List<Groups> groupSeed**
    - **Battles 1-10, 11-20 and so will be made here**

## Party List Script

* **Holds all our characters**
  + **Serialize Field Groups**
    - **currentParty**
    - **partyInven**
    - **roster**
    - **enemyGroups**
  + **List<GroupSeed> checkpoints(tentatively named)**
    - **This is where the enemy groups get grouped up into their…seeds. Battles 1-10 of the like**

## Shop

## BattleLog

* Battle log - history of damage/move/item usage and how much damage they do.
  + Include buffs and debuffs
  + Item usage as well
    - This person used this. 300 damage.
  + Needs to scroll
    - ATK, ITEMS, STATS, LOG, FLEE
      * Need a button and window for it
* Needed:
  + **In Unity:**
    - **Log Button**
    - **Log Container**
    - **TMP Log Prefab**
  + **In Script (will be in generator script)**
    - **GameObject**
      * **logContainer**
      * **logPrefab**
    - **List<string> logList**
  + Function AddBattleLog(string newLog)
    - Shows the log of attacks, heals, and such
    - To do:
      * Add the newLog to the list
      * Have the y position made
      * Go through the logList and instantiate them in the logContainer
      * Space them out
      * Have the log’s text = the message
    - Execution
      * logList.Add(newLog)
      * Have GameObject logObj instantiate the logPrefab in the logContainer
      * Separate by yPosition
      * logObj.GetComponent<RectTransform>().anchoredPosition = new Vector2(0f, -yPos);
      * textObj.GetComponent<TextMeshProUGUI>().text = newLog;

## C

**Can be used as events**

1. **Character Level Up**: This event could be triggered when a character gains enough experience to level up. The event could update the character's stats based on their StatsList.
2. **Character Move Used**: This event could be triggered when a character uses a move from their Movesets. The event could handle the effects of the move, such as dealing damage, applying status effects, etc.
3. **Character Animation Change**: This event could be triggered when a character's animation needs to change, such as when they start moving, attack, get hit, etc. The event could change the current animation to one of the charaAnimations.
4. **Character Sprite Change**: This event could be triggered when a character's sprite needs to change, such as when they equip a new item, take damage, etc. The event could change the charaSprite to a different sprite.
5. **Character Stat Change**: This event could be triggered when any stat in charaStatList changes. The event could update the display of the character's stats, check if the character's HP has reached 0, et

The execution of attacks in game development can vary greatly depending on the type of game, its complexity, and the specific requirements of the game mechanics. However, there are some common patterns that many developers use.

In the context of your BattleSystem.cs script, you're using a common approach where you have a switch statement that handles different types of moves. This is a good approach because it allows you to easily add new types of moves in the future.

Here's a simplified version of a common approach to executing attacks:

1. **Determine the type of attack**: This could be a basic attack, a special ability, a magic spell, etc. In your case, you're determining this based on the MovesType of the move object.
2. **Calculate the effect of the attack**: This could involve calculating damage, healing, status effects, etc. You're doing this with your damageCalc object.
3. **Apply the effect to the target**: This could involve reducing the target's health, applying a status effect, etc. In your case, you're storing the results of the calculation (damage, healing, etc.) to be applied later.
4. **Trigger any additional effects**: This could involve playing a sound or animation, triggering a counterattack, etc. In your case, you could use your AttackEvent to trigger these additional effects.
5. **Check for end of turn or end of battle conditions**: This could involve checking if the target's health has reached 0, if all actions for the turn have been executed, etc. In your case, you're clearing the actions for the next turn at the end of your method.

This is a very general approach and the specifics can vary greatly depending on the game. For example, in a real-time game, you might not have turns and actions might be executed immediately when the player presses a button. In a more complex turn-based game, you might have a more complex system for determining the order of actions, calculating effects, etc.

Correct, if you use a Coroutine in Unity, the actions within the Coroutine can be spaced out over time. This is often used for animations, timed events, or any sequence of actions that you want to happen with delays in between.

In your OnAttack method, if you want to space out the actions (like playing animations, applying damage, showing messages, etc.) over time, you could use a Coroutine. Here's an example

public IEnumerator OnAttack(CharacterTemplate user, Moves move, CharacterTemplate target)

{

// Character loses mp

user.TakeMP(move.MPCost);

yield return new WaitForSeconds(1f); // Wait for 1 second

// User animation plays

PlayAnimation(user, move.Animation);

yield return new WaitForSeconds(move.Animation.length); // Wait for the animation to finish

// Target pain animation plays

PlayAnimation(target, move.PainAnimation);

yield return new WaitForSeconds(move.PainAnimation.length); // Wait for the animation to finish

// Target takes damage, so the health bar goes down

target.TakeDamage(move.Damage);

yield return new WaitForSeconds(1f); // Wait for 1 second

// If buffs or debuffs are applied, they are applied

ApplyBuffsDebuffs(user, move, target);

yield return new WaitForSeconds(1f); // Wait for 1 second

// Words describing what just happened appear on the screen

ShowMessage($"{user.Name} used {move.Name} on {target.Name} for {move.Damage} damage!");

yield return new WaitForSeconds(3f); // Wait for 3 seconds

// Seconds pass

yield return new WaitForSeconds(1f); // Wait for 1 second

}

* Have 5 “growths” of stats: Excellent, Great, Good, Bad, Worst (enum Growths)
  + Make each stat one of those (could need a new variable, i.e, efficiencyGrowth(?))
* When leveling up, go through each stat and see what those stats are.
* Have the stats roll in accordance of their growths
* For SQL

If else (case)

Partition

Group by

Having clause

All the joins

## Between Fights

* The nonfighting zone. When a session is beaten, player will be transported here. Used to prepare for the next battle (equip, shop for new materials, save, quit, etc)
* Menu buttons:
  + Party
  + Items
  + Shop
  + Next Battle
  + Quit
  + Options
  + Exit

## Party

* Used for players to switch, add, remove, or equip items to the party members in their possession.
  + Note: equipment will stay on the character when equipped, whether they are in the party or not.
* Needed:
  + Generate roster of characters player owns
    - Probably gray out or say “equipped” characters that are already in the party.
      * If rosterChara is in party, switchButton.SetActive(false)
  + Generate the current party members the player has equipped
  + On both, have options for both roster characters and current party members
    - View
      * View character’s stats, skills (?), etc
    - Switch
      * Switch roster member with party member, or party member with roster member.
    - Equip to
      * Comes with a dropdown. Equip armor 1, armor 2, or weapon of the selected character
    - Add
      * Add roster character to party (if the current party is not full)
    - Remove
      * Remove a character from the party. Set their roster photo back to normal so that you can add them again.
    - Close
      * Close menu so you can go to the other options

## Items

## Shop

## Next Battle

## Quit

## Options

## Exit

## 