**Equipment/Items:**

* I still need to make the unit inventories children of the unit in order to match the design of the equipment slots.
* make a shield class
* Make amulet/rings do something when equipped
* Make it so the dropping an item causes the item to enter the first available floor space.
* When items are stacked, the health of the resulting stack is the minimum of the 2 stacks.
  + Keep in mind, the stack health represents the health of the item on top of the stack
  + This can be used to “cheat” and repair items. This is fine as long as items where this really matters do not stack or if there is an easily accessible “free” repair interaction.

**Text boxes:**

* Make wrapping setting that toggles forced wrapping or no wrapping (in the event there are no spaces on the current line)

**Units:**

* Make a gold djikstra map and give units a greed value that determines how badly they want to get gold.
* More unit features like ranged attacks and easier spawning.
* Teams
* new variable "Unit attackTarget" that holds who we currently want to attack

**Level Generation:**

* more dungeon tiles: doors, pits, lava, water, drawbridge?, ect.
* Make random gorges across the map during generation. If they block the player, they always have the option to jump in.
* Make bridges across gorges in some themes EX. in a crypt there may be a bridge but in a cavern there wouldn't be.
* The 2 repos I have starred on github have a lot of good information
* <https://www.gridsagegames.com/blog/2014/06/procedural-map-generation/>

**Coatings:**

* RoguelikeObjects have a CoatingEnum with the following possible values:
  + Bloody
  + Oily
  + Wet
  + Poisoned
* An int CoatingAmount indicates how saturated the object is with the coating.
* An object can only have one coating at a time.
* A Clean() function will remove all coating from an object. Player can automatically do this to any object.
* An object that is coated in oil will catch on fire even if it is not flammable. An oily object is immediately engulfed in flames when lit (instead of starting slightly onFire). Every fire tick, the CoatingAmount of the oil is reduced by 1 to represent the oil burning off.
* Objects can only be stacked if they have the same Coating type.
  + When stacks are combine, the new CoatingAmount =   
    floor( (stacksize1 \* coatAmt1) + (stacksize2 \* coatAmt2) / (stacksize1 + stacksize2) )

**Fire:**

* Items that are on fire will not automatically stack when picked up. They can be manually stacked on a nonburning stack though.
  + When stacks are manually combined:  
    Max( 1, floor( (stacksize1 \* fireAmt1) + (stacksize2 \* fireAmt2) / (stacksize1 + stacksize2) ) )

**Walls:**

* Give walls a moveable tag that allows them to be moved around by players, by explosions?

**QOL:**

* everything has interacts which are functions that can be called and have string names associated with them.
  + right-clicking in the game pulls up a list of all possible interacts.
  + bumping into walls, walking onto floors, walking into units, all do the first action in the list
  + for example, a door is a wall. when you bump into the door, it does the open interaction (the first interact in the list)
  + the first interact may change depending on certain variables. For example, having a key may switch the first interact to be unlock and the second to be open

CURRENT TASK

* ~~Constrain the inventoryGUIs scroll to stop scrolling before having any empty lines.~~
* Fix the bug with InventoryGUIs where double pickupdrops will be created.
* ~~Figure out why opening multiple inventories increases the scroll amount each time. The first inv I open scrolls normally. The second scrolls 2 rows at a time and so on.~~
* ~~FIX THE MEMORY LEAK~~
  + ~~UNITY EDITOR IS NO LONGER LOSING THE RESOURCES ON EXITING PLAY MODE~~
  + ~~Healthbars were the Culprit they were repeatedly creating new sprites.~~
  + ~~Sprite.Create is the real issue. REMOVE ALL INSTANCES OF THE SpriteController and Sprites class as they are causing additional memory leak just not during update.~~
* ~~Continue refactoring the sprite stuff for: axe, dagger, armors, units, ect.~~
* ~~Rip out the old inventory code. Specifically all of the InventoryGUIEnum stuff.~~
* ~~Popup Class – extends the functionality of a GUIComponent to allow it to be “popped up” and closed. Popups will be tracks by a new PopupController class.~~
  + ~~A popup and close function can be extended by the child popup. For example, when the containerGUI is popped up, it will probably load its current inventory into the pickup drops and the title ect.~~
  + ~~As a general rule, for the sake of simplicity, any popup will be safe to close at any time. This means that closing a dialogue popup for example should always be safe.~~
* ~~PopupController – This class will hold a stack that contains all popups. Additionally, this class will contain bounds that popups cannot be moved out of (to prevent a popup from being cutoff).~~
* ~~Make it so all controllers exist in the editor before game start. This will make debugging easier and utilize the wysiwyg features of unity.~~
* Take another look at the entry animations to ensure that the timing is not suffering from some rounding error. Also remove existing animations on SetText
* Remake the healthbars.
* Fix the issues with the pit sprites. (not all of the sprites are being utilized)
* I removed the gold pathfinding map stuff in the gold class. This will need to be readded eventually
* The RoguelikeObject Inheritance rewrite:
  + Plan of attack:
    1. ~~First, update inventories to hold RoguelikeObjects instead of the old items~~
    2. ~~Update and work with the new ItemSpriteRenderer class to get inventories properly displaying RoguelikeObjects~~
       - ~~Currently, the Entry class is messing up because of the lossy scale vs local scale stuff. Try to get Entrys working with lossyscale so they can easily be integrated anywhere in the game~~
       - ~~Make sure the outline on the characters looks alright across the board.~~
    3. ~~Add a way to easily create and destroy RoguelikeObjects~~
       - ~~Move the MakeItem (MakeRoguelikeObject) function into the RoguelikeObject class~~
    4. ~~Make sure all old functionality still works:~~
       - ~~Dragging & dropping~~
       - ~~Dropping on ground~~
       - ~~Splitting stacks~~
       - ~~Ect~~
    5. THEN look at WorldObjects and getting them to work.
* The WorldObject creation:
  + Plan of attack:
    1. ~~Create the WorldObjectClass. Peek at the definition of the current wall class for a better idea of required stuff.~~
    2. ~~I am now leaning toward having each wall/unit/floor space on the map be an inventory that hold a max of one object. When something is put there using the MoveItem function, the wall will become visible and active.~~
    3. ~~Make a wallController which has inventories for each wall space. Adding placing a wall adds it to the inventory and makes it show. Essentially, everything is moving by jumping from inventory to inventory.~~
    4. ~~Update the class diagram with the new wall stuff. Update the new code with comments.~~
    5. ~~Remove the old Character Containers.~~
    6. ~~Remove the old wall code and begin replacing it with the new wall code. Test a lot during this step to ensure nothing breaks.~~
    7. ~~Begin looking at the requirements for making a connectedWall object.~~
    8. ~~Go back and ensure the class diagram is up to date.~~
* Floor Creation:
  + Plan of attack:
    1. ~~Create the Floor2.cs~~
    2. ~~Create the FloorConnected.cs~~
    3. ~~Create the FloorController.cs~~
* Unit Creation:
  + Plan of attack
    1. ~~Move units over to the new system.~~
    2. Refactor the existing Unit code. You may want to move some of the Unit functions up to the WorldObject or RoguelikeObject classes.
    3. Perform tons of testing.
    4. Push changes!!!
    5. Delete the old walls, floors, units, and players. Remove the 2’s from all of the new items
    6. Do a huge comment/class diagram sweep ☹

NEW KEYWORD IN CODE

[REMOVE?] – add this as a comment to code that may need to be removed in the future.  
 [NEEDS WORK] – This function will need to be updated in the future

Add Item:

* Takes an item and adds as much of it as possible to an inventory.
* Returns how much of the item was added.

MoveItem:

* Takes an item, a desitination inv, and optionally an amount. If the amount is greater than the item.StackSize or = -1, move the entire item over. Else, create a new stack of size amount and add as much of it as possible to the other inventory, then reduce the size of the original stack by the amount moved and delete the rest.