*For better readability make sure that View->Print Layout is* ***NOT checked***.

**Index and Links**

#35 Inventory Page
#34 Utilize Str/Int/Char
#33 Level-up HP (and Str/Int/Char)
#32 DayScreen Summary Lists
#31 Karma, Influence, Butterfly Effect (KIB)
#30 Score.
#29 Make selected house and transit "scroll into view" when selected item is "hidden".
#28 Alert user when house % or transit item % goes below zero and is removed.
#26 Change mouse pointer.
#25 Audio.
#24 Overthrow Trump.
#22.3 Show "map" of player's current city.
#22.2 Show "map" of locations with associated events and player's current "Home".
#22.1 Move locations.
#22 Events only affect certain regions.
#21.1 Add jobs.
#21 Add lots more items.
#20 Add lots more events.
#19 Add width to self.body = { … } to dynamically change main text box width.
#16 Manual character creation.
#9 Counteracting the effects of events. (Too difficult for gameplay?)
#8 Low or zero sanity.
#7 eod_mods Summary Screen (enhanced StoryScreen)
#6 eod_mods
~~~~~~~~~~~~~~~~~~~~~~~~~~
########## Done ##########
~~~~~~~~~~~~~~~~~~~~~~~~~~
#11 Add un-process events.--Done
#27 Change to dynamic sizing of the window.--Done (partial: F11 appears working)
#10 Add process events in eod_mods(). --Done
#23 Change HP font size.--Done
#X Make hours not go negative on work.--Done
#20.1 Refactor events into events.py.--Done
#18 Refactor jobs import to make uppercase.--Done
#17 Add story_text to events.--Done
#3 Take off hours for driving to work. Depends on (#2: Job and Jobs class).--Done
#4 Buff events rather than add duplicate events.--Done
#5 Make EventsScreen surface bigger to fit more events.--Done
#14 Killing the character again not fully working.--Done
#8 Font color for main box needs fixing.--Done
#13 Only allow Job.work() to work the number of hours remaining in the day.--Done
#12 Make game_state.game.mod_hours( ) function and refactor code. --Done
#15 Add character.job.income to bonuses_by_ratio.--Done
#1 Staying with Friends: Distance to stores change ...randomly.--Done
#2 Add a job class for character.--Done
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
########## Other Ideas ##########
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#998 Genre
#999 Different screens flow diagram (state machine? or...?)

**Features Under Consideration**

- Idea: Use "W" and "S" for "Up" and "Down" on the main menu and "D" as "Enter".
  - Good for touchpad users.
- Add various "characters".
  - https://www.reddit.com/r/gaming/comments/68ij9r/ok_genius_idea_for_a_game/
  - Trump, Clinton, Bush, …, …, A. Lincoln, G. Washington, …, …,
  - How to integrate?
  - Boss battles?
  - Buyable "Heroes"?
- TODO: After walking 25 miles +0.1 HP Upgrade?
- Event months remaining:
  - Idea: All inactive events go down one month remaining each month.
    - Eventually they go away.
  - Idea: Add (#) beside each event to show how many months they have remaining.
    - Ex: You got an STD (3). Meaning 3 months remain.
- Make an alert if there are still day hours remaining and the user presses "Next Month".
  - Just in case they do it on accident.
  - Go to next month? Yes, No.
- On CharacterHUD items list, only show important items.
  - No need to show transit and housing items as they are in the other lists.
  - Show: Food, Clothing, Cash, First Aid Kits
  - Create *Inventory Page* to show everything else, such as gardens, seeds, etc.
- Idea: HP living reward
  - Each day HP goes up +0.1 until maxxed.
- Age, Gender
  - How to utilize these?
  - Age: …
  - Gender: …
- Mini-games.
  - Snake Game.
- After activating an event on EventScreen, it goes back to DayScreen.
  - Optional: Have it stay on EventScreen.
- Text to show when there are no events?
  - Idea: "There are no events to display."
- Bug: Food going negative on store inventories.
- Idea: Implement A.I. Q-Learning somewhere.

**Small Things Done**

- Done: ~~Add +1 sanity each time you go to work.~~
- Done: ~~Do not allow both lists on EventScreen to be selected at the same time.~~
  - ~~De-select the list that was clicked oldest.~~
- Fixed: ~~Bug: On EventsScreen: After "Use first aid pack", if still alive, the activated event is still shown on left side. It is activated but lists are "stale". Fix: Redraw PygameUI elements if user is still alive (def click_use_first_aid). See draw_store_lists( ) for similar.~~

- Done: ~~Character HUD main lists: Change bgcolor of selected to bgcolor of game, so it is "not selectable".~~
- Done: ~~Document code for Friday's code review. Every class and many functions.~~

Programming strategy

**#35 Inventory Page**

- Use items (eg. gardens+seeds), which is different from counteract events with items.
    - Lottery tickets
    - Gardens+Seeds
    - …
- This will be about the sixth menu item on DayScreen. So. That means probably making some code for the Menu class which automatically wraps into *two columns* when there are more than five or so Menu items.
    - Another option for this would be to add *Up and Down arrows* beside the menu items when there are more items than are displayed. And then adding code so that when a menu item is "off screen" it "scrolls up".

Programming strategy

**#34 Utilize Str/Int/Char**

Strength, Intelligence, Charisma

- How to utilize?
    - Intelligence:
        - Counteract -sanity.
        - Ex: Character has 4 intelligence points to start the day. There is a curfew event: -2 sanity. Character now has 2/4 intelligence for the day but no loss of sanity.
    - Charisma:
        - Counteract -cash.
        - Ex: Character has 2 charisma points to start the day. There is Puppies: -$1000. Character now has 1/2 charisma for the day but no loss of cash.
    - Strength:
        - Counteract -HP.
        - Ex: 4 strength counteracts up to -4 HP, once per day.
- At the end of each day these are refilled.
    - So Str: 5/5, Int: 3/3, Char: 1/1.
- What about spell casting? XX requires XX intelligence to cast.
- Some events may knock down player's Str./Int./Char.

Programming strategy

**#33 Level-up HP (and Str/Int/Char)**

HP, Strength, Charisma, Intelligence.

- How to level up?
    - Each time an event (good or bad) is passed there are bonuses.
    - 0.1 bonus per event which is passed.
    - Let the user decide where to level up / spend upgrade points.

Programming strategy

**#32 DayScreen Summary Lists**

Ongoing summary of DayScreen events (Today: Notice, Tomorrow: Warning).

- PyGameUI Lists (aka Housing and Transit).
- Put in bottom left corner of DayScreen.
- Lists are updated throughout the "Day" as the user takes actions.
- Idea: On select, show a "Hint" for the notice.
    - Ex: "Go buy some food at the store."
- Today's Events (Notices):
    - A List to show today's events:
        - Notice: You biked to work: K+1, I+1, B+1 (-1.4% bicycle)
        - Notice: You drove to work (40 kg $CO_2$): K-1, I+0, B-1 (-1.2% car)
        - Notice: You boated to work (xx $CO_2$): ...
        - Notice: You sailed to work (0kg $CO_2$): K+1, I+1, B+0 ...
        - Notice: Event ___ was activated: +1 HP, +$1,000
        - Notice: Worked X hours, made $1,000: K+1, I+1, B+0; Sanity+1
- Tomorrow's Events (Warnings): --Partially done
    - A List to show upcoming events.
    - Warning for eod_mods (tomorrow).
    - Idea: Make warn icon for this.
        - Warn: Food is 0, you are going to lose health tomorrow!
        - Warn: Staying with friends, you are going to lose a sanity tomorrow!
        - Warn: Sanity <= 0, you are going to lose one HP tomorrow!
        - Warn: Clothing <= 0, you are going to lose a sanity tomorrow!
        - Warn: Events >= 5, one random event is going to be activated tomorrow!

Programming strategy

**#31 Karma, Influence, Butterfly Effect (KIB)**

- KIB
    - Idea: A Karma - Influence - Butterfly Effect menu screen.
    - Karma:
        - You did some nice things. Now nice things happen.
            - Let's say Karma reaches +4
            - Random good event is created.
            - Karma = 0.
        - You did some not nice things. Now not nice things happen.
            - Let's say Karma reaches -4
            - Random bad event is created.
            - Karma = 0.
        - Random event generation goes in eod_mods?
    - Influence:
        - Influence those around you.
            - Extreme Pollution: You convince others to reduce polluting actions.
            - Tax Collector: You have influence over the tax collector.
            - Influence reaches +4: Store purchases are 10% off.

            - Good events are +10%.
        - Influence reaches -4: Store purchases are 10% extra.
            - Bad events are -10%.
- Butterfly Effect:
    - https://en.wikipedia.org/wiki/Butterfly_effect
    - BE is all about chain reactions.
        - It is about randomness.
        - It is about a very small choice now popping up as something much larger later on.
        - Small life event leads to large life event.
    - Going to the store starts a chain reaction that results in …
    - Buying a lottery ticket starts a chain reaction that results in …
    - Let's say BE reaches +4.
        - -1 month remaining on all "negative" events.
        - +1 month remaining on all "good" events.
        - BE = 0.
    - Let's say BE reaches -4.
        - +1 month remaining on all "negative" events.
        - -1 month remaining on all "good" events.
        - BE = 0.

<u>Programming strategy</u>

**#30 Score.**

- -1000 for each inactive event --Done
- +100 for each $10000 in cash --Done
- +500 for each day and +20000 for each term --Done
- +100 for each pie
- …
- …

<u>Programming strategy</u>

**#29 Make selected house and transit "scroll into view" when selected item is "hidden".**

Selected house and transit should "scroll into view" when selected item is "hidden". That is, the user has to scroll down in the housing or transit box to see the item that is currently selected.

This needs to happen each time CharacterHUD is redrawn.

<u>Programming strategy</u>

**#28 Alert user when house % or transit item % goes below zero and is removed.**

Alert the user that transit and housing have been reset. May need some testing.

Examples (alert user in these cases):

1. eod_mods: house may be used up.

2. going to work: transit may be used up on the way.
3. going to any store: transit may be used up on the way.

Programming strategy

**#26 Change mouse pointer.**

For PygameUI items that are clickable such as the lists and alert buttons, change the mouse cursor to something else, like a "hand".

In general, the mouse cursor could be changed to something else. But what?

Programming strategy

**#25 Audio.**

1. Add one track --Done
2. Choose from tracks in a folder
3. Sound effects
4. Options menu on/off --Done
5. Options menu select music
6. Options menu change volume

TODO: Get CPU to be idle when playing music. (Impossible?)
Music sites: soundcloud, youtube, freesound.org
~~Consider pygame.mixer.music.load (http://nullege.com/codes/search?cq=pygame.mixer.music.load).~~
~~Per screen basis? On Event basis? Categorically (start menu, create character, storyscreen, day)?~~

Programming strategy

**#24 Overthrow Trump.**

See weekly scrum document for some details on this.

Programming strategy

**#22.3 Show "map" of player's current city.**

Something like a grid layout. Cities are something like 20x20 miles. Or 40x40 maybe. Maybe have a grid where each 1x1 mile is a box.
Has icons for:

- stores
- player's houses
- player's job
- other jobs in the city
- houses available for sale (?)
- current location of "friend's house"

Programming strategy

**#22.2 Show "map" of locations with associated events and player's current "Home".**

This is a USA map.
Shows a list of events (active and inactive) in "region".
Maybe shows an overview of jobs and stores in each region.

<u>Programming strategy</u>

**#22.1 Move locations.**

Is there a brainstorm of this somewhere? User may choose to move character to a new location. It costs 16 hours. The new location must be connected to the current location. That is, it is possible to move from NYC to Washington D.C. in one turn but not from NYC to LA in one turn.

We were maybe chatting about it in Facebook chat.

<u>Programming strategy</u>

**#22 Events only affect certain regions.**

1) A tsunami only affects coastal regions.
2) Each month a random event probably occurs in each region. ~~But it is probably easier to just generate eight events and dole them out (for each of let's say eight regions).~~ Easy: If the event does not affect the region then it is simply is deleted.
3) And if the event is not in the player's current location then maybe a message: ………... ?

<u>Programming strategy</u>

**#21.1 Add more jobs.**

More jobs.

<u>Programming strategy</u>

**#21 Add more items.**

More items.

<u>Programming strategy</u>

**#20 Add more events.**

More events.

<u>Programming strategy</u>

**#19 Add width to self.body = { … } to dynamically change main text box width.**

Make width an optional attribute of self.body = { … }. Then check for it in EventsLoop. Something like:

```
if hasattr(cm.body, 'width') == True:
    w = cm.body['width']
else:
    w = cm.scene.frame.w - 160 - 160
```

Then test it by changing one of the main box widths, such as character confirm width. (Before it was 300.)

Programming strategy (Not fully fleshed out)

**#16 Manual character creation.**

A number of screens ought to work. Each screen has e.g. "Go Back" and "Continue".

*First screen: Attributes placement and name*
First screen is name and attributes placement. For attributes maybe consider a "number" field (PygameUI has the List field which ought to work for this).
Draw out a layout of this screen and then implement it.
*Also note*: On this screen game_state.game.character = Character() called for the first time.

*Second screen: Location*
Choose a location. A list.

*Third screen: Choose job*
Choose a job. A list.

*Fourth screen: Visit a store. (Reach goal)*
Quick and dirty is: Copy and modify code from StoreScreenSelect.

Programming strategy

**#9 Counteracting the effects of events. (Too difficult for gameplay?)**

The item.counteract( ) code at the very end will need some analyzing. But otherwise this todo is ready to be implemented. (See 3/20 To-Do for details on counteract.)

Add to the "n" dictionary in items.py.

```
{
...
        'counteracts':
        {
                'Gun': { 'Some Event': 1,  'Another Event': 2 },
                'Dead Fish': { 'Trump Tries to Kiss You': 3 },
                ...
        }
...
}
```

When adding Gun and Dead Fish add them to the other parts of the dictionary, as necessary. Add a price, resale value, etc. Also, consider adding some other items to the counteracts dictionary (e.g. Food?).

Start with making a function to counteract. It goes in Event.

```
def counteract(self):
    '''List the items in the character's inventory that possibly
    counteract this event. This list is in PygameUI.List( ) format.
    That is, [ dictionary_item, dictionary_item, ... ].
    :return: Returns a list of items or an empty list if no items counteract.
    :rtype: list.
    '''
    # Strategy:
    # Loop through all of character's items, checking for this events.
    i = game_state.game.character.inventory.items
    temp = [ ]
    for item in i:
        list_events = ITEMS.n[ 'counteracts' ][ item.item_type ]
        if event.event_text in list_events: # This item counteracts the event!
            temp.append({
                'item': item
                'value': item.item_text
            })
    return temp
```

Add a menu item on EventScreen called "Counteract Event".

Add a warning message in the section EventScreen.__init__( ):

```
self.warning_no_items = "Your inventory does not contain any items that counteract this event"
```

Add a function for doing the counteract alert in EventScreen:

```
def counteract_alert(self):
    choices = self.selected_event.counteract( )
    if len(choices) == 0:
        self.alert(self.warning_no_items, ["OK"])
    else:
        self.alert("Here are the items that counteract"+self.selected_event.event_text,
            ["Cancel"],
            None, # No callback is necessary.
            choices,
            self.click_choices)
```

Add to EventScreen.process_before_unload( chosen_position ).
If chosen position is "Counteract Event" then:

1. Check that user has pressed on an event: self.selected_event.
    a. If no, alert( ) with an "OK" button. Return False.
    b. If yes, check that the self.selected_event is currently active (self.selected_event.activated).

i. If yes, this is where all of the character's items that counteract the event will listed. Call self.counteract_alert().

```
self.counteract_alert()
return False
```

ii. If no, do an alert with an "OK" button saying the event is not active. Then return False.

```
…
return False
```

Add *click_choices* to EventScreen:

```
def click_choices(self, selected_index, selected_value, selected_item):
    selected_item.counteract( self.selected_event )
    # Bring up an alert now that the event was counteracted.
    self.alert( ('The event was counteracted by ____ .\n'+
        'Would you like to try to counteract this event again?'),
        ['Yes, please!', 'No, that's OK.'],
        self.click_go_again)
```

Add *click_go_again* to EventScreen:

```
def click_go_again(self, confirm):
    '''If yes, go back to counteract list. Otherwise, do nothing.
    '''
    if confirm:
        self.counteract_alert()
```

Add *counteract* to the Item class.
Consider some chance code (See 3/20 To-Do for some ideas on counteract.)

```
def counteract(self, event):
    print 'Debug:Counteracting:', event.event_text
    # Add some (chance) code here to modify the event…

    print 'Debug:Using some of this item...', self.item_text
    # Add some code here to decrease item.amount or item.remaining_use.
```

<u>Programming strategy</u>

**#8 Low or zero sanity.**

Low or zero sanity.

- Idea: With low sanity [X happens] and 0 sanity [Y happens].
- Such as: Events modification.
    - "Your sanity is low! So even though you won the lottery, the cashier tricks you into giving her half of the lottery reward!"

    - "Your sanity is zero! You found a supply cache, but you thought the supplies were baby seals. You brought the baby seals to the nearest lake and now they've swam away!"
    - "Your sanity is low! A Zombie Apocalypse is occurring (this month). Its effects are double because you think the zombies are trees and you are trying to climb them."
    - Goes into event.process()?

<u>Programming strategy</u>

**#7 eod_mods Summary Screen (enhanced StoryScreen)**

How about an eod_mods summary screen? Is it useful to inform the player that some important stuff has happened? It could have a list that the user clicks to see each associated story.

This could all be achieved by adding two lists to Story Screen:
1. <u>Left list</u>: End of day mods list. Click to display the story.
2. <u>Right list</u>: Active events. Click to display story.

Items for left list:
1. Title: No Food! : The pantry. It is empty. Because you do not have any food this month, your health has gone down.
2. Title: No Clothes! : There is a saying, "Clothes make the man." Because you do not have any clothes left, your sanity has gone down. (There is also a film called Clothes Make the Man *(1915)*, based on a novella by Gottfried Keller *(1874)*.)
3. Title: Staying with Friends : It is hard not having a place to call home. Because you are staying with friends, your sanity has gone down.
4. Title: {Random Event} Occurred : Time scheduling is an important part of life, especially when trying to stay out in front of life's myriad events. Because there are more than five events in the events queue, one of them ({Random Event}) randomly went into effect!
5. Title: Stores re-stocked : Check out the new merchandise! XXX's: Come on in! We've got a lot of new items this month, such as [random re-stocked item][$price]!!! YYY's: Come on down to YYY's! You'll want to see our brand new [...random…][$price]! ZZZ's: Sunday! Sunday! Sunday! For a limited time only: [...random...][$price]!
6. … ?
7. … ?
8. … ?

<u>Programming strategy</u>

**#6 eod_mods**

eod_mods
- If mods cause health<=0, do first aid kit functionality (see EventScreen).[Done]
- If mods cause sanity<=0, sanity resets to 5 and hp-=1.[Done]
- If staying with friends, sanity-=1.[Done]
- If no clothing, sanity-=1.
- If no food, HP-=1 (already implemented).
- If len(inactive_events)>= 5, select a random event to activate.[Done]
- Stores re-stock inventory items.

- Process events (**#10**)[Done]
- … … ?
- … … ?
- … … ?
    -

Programming strategy

~~~~~~~~~~~~~~~~~~~~~~~~~~

Below here is done

Programming strategy

**########## Done ##########**

Below here is done

Programming strategy

~~~~~~~~~~~~~~~~~~~~~~~~~~

Below here is done

Programming strategy

**#11 Add un-process events.**--Done

Event unload functionality. Add a dict of effects to take place when the event is over. When no events then pass an empty dict. For example, { 'income':5000 } to have income go up 5000 when the event is over. In Event.process( ) after the months_remaining-=1, process this event_over dict if months_remaining<=0.

Programming strategy

**#27 Change to dynamic sizing of the window.--Done (partial: F11 appears working)**

Kind of a challenge. Maybe take the size window we currently have and then divide it by the "full screen size" of the user's computer screen? This is then the "stretch" ratio. Then multiply all measurements in the game by this ratio. Note: There would be separate stretch-X and stretch-Y ratios.

Programming strategy

**#10 Add process events in eod_mods(). --Done**

For each event in game_state.game.events.active_events (loop), call event.process().
Now, go update event.process(). At the bottom, when months_remaining <= 0, it needs to remove itself from events.active_events. However...

Removing an event from active_events while looping through active_events will not work.

*Thus:* Rather than removing the event from active_events, just set the event:*self.activated=False*. And then, after looping through all active_events, call a function to re-build the active_events list. This ought to go in the Events class. How about calling it *def generate_active_events( )*? Basically, create a temp list. Then loop through active_events and check if *event.activated==True*. If it is, then add it to the temp list. Then set *self.active_events = temp*.

Programming strategy

**#23 Change HP font size.--Done**

Idea

**#X Make hours not go negative on work.--Done**

… How about… Since hours worked is random… If you get to work and the number of hours you work (random) ends up being more than the number of hours left in the day… Then your boss says… "Sorry, Linda! We do not need you today.\nWould you like some pie before you go?" … in an alert box of course, with the two buttons "Sure thing!" and "Did someone say pie!?".

Programming strategy

**#20.1 Refactor events into events.py.--Done**

Move Event and Events class into events.py. Then: *import events as EVENTS*. Then in Game class: *self.events = EVENTS.Events()*.

Programming strategy

**#18 Refactor jobs import to make uppercase.--Done**

Such as uppercase "ITEMS" instead of "items". It seems more readable…
So just: import jobs as JOBS. Then change all relevant instances of jobs in the code to JOBS.

Programming strategy [done]

**#17 Add story_text to events.--Done**

…

Programming strategy

**#3 Take off hours for driving to work. Depends on (#2: Job and Jobs class).**--Done

One: In *Job.__init( )*, add *self.coordinates = { }*. --done
Two: From Store copy two functions to Job: *def distances()* and *def distance_from_house()*
Then add a call to *self.distances( )* at the bottom of *Job.__init__( )*. Change *distances* and *distance_from_house* slightly to use *self.area* rather than *self.store_location*. --done
Three: Now there is a job x,y coordinates and a character's current x,y coordinates.

- It all takes place in the *process_before_unload()* function for the DayScreen.
  - The function *Menu.process_before_unload()* is called when a user hits the Enter key on a menu item. It has the power to keep the screens from changing by returning False. It must always return True (or itself be False) in order for the screens to change.
- Go to StoreScreen in the code and copy *process_before_unload()*. Also copy the function immediately following this called *click_no_change()*. --done
- Go to DayScreen. At the bottom of the class paste *process_before_unload()* and *click_no_change()*. --done
- There is the argument for *process_before_unload* called *chosen_position (int)*.

- If chosen_position is "Work" (chose_position=3?) then validate work travel. Use the *Job.distance_from_house()* from above.
- If there is not enough time left, then use the *self.alert()* function that is in the code already to make a warning, something like "Warning: There are not hours left to make it to work!\nIt takes __ hours but there are only __ hours remaining." Then return False. --done
- If there is enough time to travel, then go ahead and alter *current_day.day_hours*, call *inventory.use_transit()*, and call *character.earn_money()*. And then return True. --done

Programming strategy

**#4 Buff events rather than add duplicate events.**--Done

Buff remaining months. (Buffing bonuses would be a similar strategy.) Then change the StoryScreen to reflect if event is new or buffed.

One: This takes place in Events.random_event(), just before appending to self.inactive_events. Before appending, loop through self.inactive_events. These are Event instances. Two of the same type of Event instance will not be the same. So it is not useful to compare them. But it is useful to compare the Event.event_text, which is the event title. So loop through self.inactive_events and look for a matching Event.event_text. If it is there, then … existing_event.month_remaining += new_event.months_remaining; and break. If the new event is not in inactive_events then go ahead and append new_event to self.inactive_events.

Some thoughts:
Thought.One: Consider for a moment… What if the event is currently "active"—that is, if it is in the list self.active_events? Should it add to inactive_events? Or buff the active event?
For simplicity, how about just adding it to inactive_events for now...
Thought.Two: Consider for a moment… What is going on with the Newspaper (StoryScreen) during this process?
*__init__* : game_state.game.events.random_event()
*gen_date* : "life will never be the same after…"+inactive_events[-1].event_text.
Thought.Three: On buffing bonuses… Not too different than buffing months_remaining. Although it probably ought to be one or the other. Curfew is twice as long AND double strength? Er, maybe no. Curfew is twice as long OR Curfew is double strength? Better. But how to decide?

Two: And thus…
It is probably best to create something like Events.last_random_event. This ought to go in Events.__init__ and be set to False initially. Then each time random_event() is called, set self.last_random_event to a dictionary: {'event' : the new_event instance or the existing_event instance, 'buffed': True or False}. Set buffed to True if it is an existing event and False if it is a new event.

Three: Then go to *gen_date* in order to change inactive_events[-1].event_text. It is now game.events.last_random_event['event'].event_text. But what is important is that the event is sometimes an existing event that has been buffed… Keeping it simple: How about the word "More"?

For example:

| | |
|---|---|
| (new): | life will never be the same after… Puppies!! |
| (buffed): | life will never be the same after… More Puppies!! (??) |

| | |
|---|---|
| (new): | life will never be the same after… Extreme Pollution! |
| (buffed): | life will never be the same after… More Extreme Pollution! (??) |

| | |
|---|---|
| (new): | life will never be the same after… You Won the Lottery! |
| (buffed): | life will never be the same after… More You Won the Lottery! (??) |

Hm… How about "Again"?

| | |
|---|---|
| (buffed): | life will never be the same after… You Won the Lottery (Again)! |
| (buffed): | life will never be the same after… Puppies (Again)! |
| (buffed): | life will never be the same after… Extreme Pollution (Again)! |

Again seems more functional. In any case… choose a way. The story text needs to change based on whether game.events.last_random_event['buffed'] is True or False. If game.events.last_random_event['buffed'] then … Else ...

Programming strategy

**#5 Make EventsScreen surface bigger to fit more events.**--Done

Well, it would be nice to see active events as well. How about combining active and inactive events on EventsScreen? Something like...

| | | |
|---|---|---|
| Inactive Events list | Story Text | Active Events list |
| | Activate Event<br>Go Back to Day | |

This is pretty straightforward and should not be too much change over what already exists. On clicking the InactiveEvents list it will change Menu.body = { … } so that the body text is updated. It also ought to do this on clicking the ActiveEvents list.
Then when the user wants to activate an event (assuming it is currently clicked), hit the EnterKey on "Activate Event". Otherwise the user could also just go back to the day.

One: By default when the user first comes here, maybe no event ought to be selected... This helps keep it simple. --done

- Change EventScreen body text to be: "Select an event to continue...".

Two: Change self.keypressArray and self.titlesArray. --done

- keypressArray = [ DayScreen, DayScreen ] is okay.
- titlesArray = [ 'Activate Event', 'Go Back to Day' ]

Three: This is in preparation for parts Four and Six. --done

- In Event.__init__(), add self.selected_event = None.
- In Event.__init__(), add some warning messages:

```
self.warn_no_event = (
        "Warning: No event selected!\n"+
        "You must select an event to activate!")
self.warn_ask_health_pack = (
        "Warning: You are about to die!\n"+
        "Would you like to use some health packs?")
self.warn_ignore_health_pack = (
        "Warning: Why did you choose not to use a health pack!?\n"+
        "Now you died!")
self.warn_no_health_pack = "Warning: No more health packs!\nYou died!"
self.warn_event_active = "Warning: The event is already activated!"
```

- Make the function EventScreen.click_event_list(), which will update the body text when a list item is clicked:

```
def EventScreen.click_event_list(self, selected_index,
        selected_value, selected_item):
        self.selected_event = selected_item
        self.body['text'] = "Oh, wtf!? \n" +selected_item.story_text
```

  - Optional: Add information on the bonuses of the event to body.
  - Optional: Add information on the months remaining of the event to body.
  - Optional: Add information on which items counteract the event to body.

Four: Make a PygameUI.List on the left and right. These will be for listing the events.
--done
Code to add:

```
g = game_state.game.events

x = PygameUI.List(g.show_inactive_events(), (200, 224, 200))
x.frame = pygame.Rect(4, 4, 150, Menu.scene.frame.h -8)
x.frame.w = x.container.frame.w
x.border_width = 1
x.container.draggable = True
x.callback_function = self.click_event_list
self.scene.add_child(x)

x = PygameUI.List(g.show_active_events(), (200, 224, 200))
x.frame = pygame.Rect(Menu.scene.frame.w -154, 4, 150, Menu.scene.frame.h -8)
x.frame.w = x.container.frame.w
```

```
x.border_width = 1
x.container.draggable = True
x.callback_function = self.click_event_list
self.scene.add_child(x)
```

- At the above code to the end of EventScreen.__init__().
- In Events there needs to be two functions to create the items in these lists.

  (*Note that PygameUI.List is modified—see Note 3.*)

  Events.show_inactive_events(self):
  Events.show_active_events(self):

```
def show_inactive_events(self):
    temp = [ ]
    for event in self.inactive_events:
        temp.append( {'item':event,'value':event.event_text} )
    return temp

def show_active_events(self):
    '''Same as inactive but uses self.active_events.'''
    …
```

  - Note 1: A direct reference to the event during onclick callback is helpful.
  - Note 2: Each function returns a list of dictionaries:
    [ {'item':EventInstance,'value':EventInstance.event_text}, ... ]
    "item" is a direct reference to the event instance.
    "value" is the text displayed in the list.
  - Note 3: PygameUI.List is changed. It used to take a list of strings. Now it takes a list of dictionaries.

Five: This is in preparation for part Six. --done

- In EventScreen add two functions:

  EventScreen.click_use_first_aid()
  EventScreen.click_died()

```
def click_use_first_aid(self, confirm):
    '''User clicked "Use first aid packs" or "Do not use first aid packs".
    If "Do not use" end the game. Else try using first aid packs
    until there are no more remaining or until the character's health > 0.
    :param boolean confirm: True if "Use" is pressed, False if "Do not" is pressed.
    '''
    c = game_state.game.character
```

```
        if confirm is False:
            self.alert(self.warn_ignore_health_pack, ['OK'], None, self.click_died)
        else:
            n = 0
            while c.health <= 0 and c.inventory.use_item(Item('First Aid Kit'), 1) is True:
                c.health += 1
                n += 1
            If c.health is still <= 0:
                self.alert(self.warn_no_health_pack, ['OK'], None, self.click_died)
            Else:
                if n == 1:
                    m = "You're still alive thanks to a health pack!"
                else:
                    m = "You're still alive thanks to those "+str(n)+" health packs!")
                self.alert(m, ['OK'])

    def click_died(self, confirm):
        '''User clicked "OK". So end the game.
        In EventsLoop this will immediately jump
        to GameOverScreen, assuming also of course
        that health <= 0.
        :param boolean confirm: Confirm is always true in this case.
        '''
        character.is_dead = True
```

Six: The EnterKey has been pressed. Last part! This part follows the chart: https://www.lucidchart.com/documents/edit/5da609db-2674-4ab0-b412-621ea7d57879#

- Find EventScreen.process_before_unload().
- Remove the existing code there. Replace it with the following. Then test the program.

```
        '''The user pressed "Activate Event".
        '''
        if chosen_position == 0:
            # Has the user clicked on a list yet?
            if self.selected_event is None: #selected_event is in Event class go do I ref that?
                self.alert(self.warn_no_event, ["OK"])
                return False # Stay on EventScreen
            # Clicked on something.
            # Is it active?
            if self.selected_event.activated is True:
                self.alert(self.warn_event_active, ["OK"])
                return False # Stay on EventScreen
            # The event is not activated.
            # Activate event.
            game_state.game.events.toggle_event(self.selected_event)
```

```
                # Check character.health.
                c = game_state.game.character
                if c.health >= 1:
                        return True # Back to DayScreen
                elif c.health <= 0:
                        self.alert(self.warn_ask_health_packs,
                                [ "Use some health packs.",
                                "Do not use any health packs." ]
                                self.click_use_first_aid)
                        return False # Stay on EventScreen
        elif chosen_position == 1:
                # User pressed Go Back to Day.
                # In this case simply return True.
                # This will go back to DayScreen.
                return True
```

~~#X Programming strategy~~ (See #6)
~~**On EventScreen when the user selects to run an event, the event fires for the first time. After the event has fired, make sure the user is still alive.**~~

Programming strategy

**#14 Killing the character again not fully working.**--Done

...(game_state.first_game_event not working).
Same as before. When killing the character, the user has to make an event (keyboard, mouse, etc.) before the character dies. The pygame event post (game_state.first_game_event) is not working...?

Programming strategy

**#8 Font color for main box needs fixing.**--Done

Yellow menu item text may need to be bolder or darker. Small text for PygameUI needs to be larger (and/or dynamically resize the screen based on screen resolution — see "Under Consideration").
…

Programming strategy

**#13 Only allow *Job.work()* to work the number of hours remaining in the day.**--Done

How does this feature sound?
This is in the Job.work() function.
If hours_worked > day_hours, then hours_worked = day_hours.
Then money_made is going to be less!

Programming strategy

**#12 Make game_state.game.mod_hours( ) function and refactor code.** --Done

This is going to be useful to validate the reduction of day hours. When calling this, if day_hours is less than zero then make it zero. When done refactor the code. Remove all instances of:

*game_state.game.current_day.day_hours* -= *XXX*.

and replace with

*game_state.game.mod_hours( XXX, operation )*

This way the hours never goes negative.

Programming strategy

**#15 Add character.job.income to bonuses_by_ratio.**--Done

…

Programming strategy

**#1 Staying with Friends: Distance to stores change ...randomly.**--Done

This all takes place during store selection. The screen for this is StoreScreen, which actually leads to StoreSelectScreen, which sounds like it is kind of in reverse. It is probably best to set the coordinates for a Friend's house at the start of each day, rather than randomly doing it every time the user goes to the Store selection screen. (If random every time they go to the selection screen then it would be easily gamed...) The Locations class is a good place for for this code.

One: Go to the Locations class. -- done

Two: In Locations.__init__(), add: --done

*self.friend_location = { 'coordinates': [ 0, 0 ] }*

Three: Add a function Locations.update_friend_location(self): -- done

```
def update_friend_location(self):
    '''This function sets the 0th and 1st element of self.friend_location
    to a random float, -20.0 to 20.0.
    '''
    self.friend_location.coordinates = [
        random.uniform(0.0, 20.0) * plus_minus(),
        random.uniform(0.0, 20.0) * plus_minus()
    ]
```

Four: In Store.distance_from_house(): -- done

Change the location of a friend's house from:

*c1 = [0,0]*

to:

*c1 = game_state.game.locations.friend_location*

Five: At the start of StoryScreen, call update_friend_location( ): -- done

*game_state.game.locations.update_friend_location()*

*#Did not implement yet#*

Extra: If job travel is implemented (#2 and #3) then update the Job.distance_from_house() function as well.

<u>Programming strategy</u>

**#2 Add a job class for character.***--Done*

Add a jobs class to handle jobs. Add jobs dictionary to a jobs.py file. Jobs ought to be locational. It also ought to be random, maybe 10-20?

Create two empty classes, Jobs and Job. These do not need to be nested classes.

*Jobs class*

- Add an empty def __init__(). -- done
- Add a function: Jobs.random_job(self). --done
- In Game.__init__(), add self.jobs = Jobs(). --done

*Create jobs.py*

- Add import jobs at the top of Trumpocalypse.py. --done
- In jobs.py make a dictionary: --done

```
# Area is one of 'urban', 'suburban', or 'rural'
j = {
        'CEO': {
                'title': CEO,
                'income': 20000,
                'company': 'United Chocolate Refiners, Inc.',
                'area': 'suburban',
                'events': {
                        "Personal Emergency": 4.0,
                        "Easy Day":8.0,
                        "Got Call as Leaving":9.0,
                        "Done Early":7,
                }
        },
        'Plumber': { … },
        ...
}
```

*Jobs.random_job(self) --done*

- Return a Job instance.
- r = random.randint(0, len(jobs.j.keys())-1)
  x = jobs.j.values()[r]
  return Job(x['title'], x['income'], x['company'], x['area'], x['events']).

*Job class --done*

- In __init__(title, income, company, area, work_events) add:
  self.title = title
  self.income = …
  self.company = …
  self.area = …
  self.work_events = …

*Job.work(self) --done*

- Returns string: A work message to display on the body of WorkScreen.

- Refactor Jesse's code from WorkScreen by adding it to Job.work().
- At WorkScreen: *game_state.game.character.job* will cause an error but it will be defined in just a moment. (See Character class below.)

```
work_text = game_state.game.character.job.work()
self.body = {
        'text': work_text,
        'font_size': 32,
        'top': 20,
        'height': 300
}
```

~~*Job.quit(self)*~~

- ~~j/k …~~

*Location class--done*

- In Location.__init__(), add:
  *r = random.randint(0, 10)*
  *self.jobs = [ game_state.game.jobs.random_job() for i in range(0, 10+r) ]*
- Add a random job function to Location:

```
def random_job(self):
        r = random.randint(0, len(self.jobs)-1)
        return self.jobs[ r ]
```

*Character class--done*

- Remove self.job = 'Plumber' and self.job = 'CEO'.
- Replace with self.job = self.location.random_job().

*Refactor Trumpocalypse.py--done*

- Find all instances of game_state.game.character.job and replace with game_state.game.character.job.title.
- Find all instances of game_state.game.character.income and replace with game_state.game.character.job.income

Programming strategy

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

--

Programming strategy

**########## Other Ideas ##########**

--

Programming strategy

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

--

Idea

**#998 Genre**

Survival game?

Idea

**#999 Different screens flow diagram (state machine? or...?)**

http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm
Make a diagram of the different screens (?) and where they go to. Certain screens must connect to the GameOverScreen, such as EventScreen, because sometimes a character may die here. Try to determine all of the screens which may lead to GameOverScreen.

- Make a new lucid chart [:hippie emoji:].
- It may be helpful to draw out the diagram quickly by hand to get an idea of what to create.
- Use the simple class diagram to work with the screen names (or get them directly from Trumpocalypse.py): https://www.lucidchart.com/documents/edit/b5f76166-d91e-4f1c-9af3-9abe836ffc69
    - Note: Be aware that the simple class diagram may be missing a few screen names…
- … ?