For extra git bonuses do a push after each sub-section's sub-sections (so 1.One, 1.Two, ...). Maybe even change to a different branch and then merge at the end. Strategies with *X)* may not be fully fleshed out.

(*Note: For better readability make sure that View->Print Layout is NOT checked…*)

Index and Links

[#22.2 Show "map" of locations with associated events and player's current "Home".](#_9l4i05sq3yp2)

[#22.1 Move locations.](#_uead4bcnl28r)

[#22 Events only affect certain regions.](#_fzhrlf4kw21i)

[#21.1 Add jobs.](#_cid9mnmua4n)

[#21 Add lots more items.](#_f1tp3yitr74q)

[#20.1 Refactor events into events.py.](#_c13nut95kgov)

[#20 Add lots more events.](#_nt1ewlamrbx1)

[#19 Add width to self.body = { … } to dynamically change main text box width.](#_b3zqw2mmgxdn)

[#16 Manual character creation. A number of screens ought to work.](#_gdz8ekzh9cn6)

[#11 Add un-process events.](#_ynp1086knrej)

[#10 Add process events in eod\_mods().](#_xr13d6ritwa7)

[#9 Counteracting the effects of events. Depends on (#5: new EventScreen).](#_5t8rl8gt9f7o)

[#X Low or zero sanity.](#_ajmtzprro5gv)

[#X eod\_mods](#_i2jbqzs153zd)

[#X Make hours not go negative on work.](#_ujr9rt9pf865)

[#X Different screens flow diagram (state machine? or...?)](#_wjn872o56ub7)

[#18 Refactor jobs import to make uppercase.--Done](#_hqdykbln3jh0)

[#17 Add story\_text to events.--Done](#_d74b2m25urgx)

[#3 Take off hours for driving to work. Depends on (#2: Job and Jobs class).--Done](#_s7eaclqoo93h)

[#4 Buff events rather than add duplicate events.--Done](#_sok0564mv7al)

[#5 Make EventsScreen surface bigger to fit more events.--Done](#_a6zjqdwzxsqw)

[#14 Killing the character again not fully working.--Done](#_hf4xuy2qmyu5)

[#8 Font color for main box needs fixing.--Done](#_5pqu3wp0ps63)

[#13 Only allow Job.work() to work the number of hours remaining in the day.--Done](#_abqymhsz3y8m)

[#12 Make game\_state.game.mod\_hours( ) function and refactor code. --Done](#_u09foa8l27t8)

[#15 Add character.job.income to bonuses\_by\_ratio.--Done](#_urghlac2zvsz)

[#1 Staying with Friends: Distance to stores change ...randomly.--Done](#_bhlmyzroxfzq)

[#2 Add a job class for character.--Done](#_h94knrskaaij)

Under consideration

* Document code for Friday's code review. Every class and many functions.
* Consider dynamic sizing for stretching the window size.
* Mini-games.
* Character HUD main lists: Change bgcolor of selected to bgcolor of game, so it is "not selectable".
* 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.
* After activating an event on EventScreen, it goes back to DayScreen. Should it stay on EventScreen? Also, what text should show when there are no events, rather than "Select an event to continue…"?
* **#X** eod\_mods
* **#X** Low or zero sanity.
* Bug: Food going negative on store inventories.
* Make main text box wider? How about width of page minus CharacterHUD lists? About Menu.scene.frame.w - 154 - 154 (left and right lists take 154 each).
* **#X** Audio tracks. Where? On per screen basis (eg DayScreen, WorkScreeen)? On an Event basis (e.g. Puppies!!)? How about categorically (start menu, create character, storyscreen, day)? Per screen probably easiest to implement, yes?  
    
  Consider pygame.mixer.music.load (http://nullege.com/codes/search?cq=pygame.mixer.music.load). Audio options on OptionsScreen()? What is on OptionsScreen? 1) Go back to StartMenu; 2) Change volume; 3) Audio on/off.  
    
  Music sites: soundcloud, youtube, freesound.org

Programming strategy

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

Programming strategy

###### #22.1 Move locations.

Isn't this a brainstorm somewhere. (....?)

Programming strategy

###### #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.

And if the event is not in the player's current location then maybe a message: ……….... ?

Programming strategy

###### #21.1 Add jobs.

More jobs.

Programming strategy

###### #21 Add lots more items.

More items.

Programming strategy

###### #20.1 Refactor events into events.py.

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

Programming strategy

###### #20 Add lots more events.

More events.

Programming strategy

###### #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 choices: "Go Back" and "Continue".

*First screen: Attributes placement and name*  
First screen is name (test and implement text box) 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() probably ought to be called for the first time.

…

Programming strategy

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

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

###### #10 Add process events in eod\_mods().

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

###### #9 Counteracting the effects of events. Depends on (#5: new EventScreen).

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.
   1. If no, alert( ) with an "OK" button. Return False.
   2. If yes, check that the self.selected\_event is currently active (self.selected\_event.activated).
      1. 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*
      2. 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.*

Programming strategy

###### #X Low or zero sanity.

Low or zero sanity.

* Maybe 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!" … Or … "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!" … Or … "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."
  + This would go in event.process() then. Right?

Programming strategy

###### #X eod\_mods

eod\_mods

* If mods cause health<=0, do first aid kit functionality (see EventScreen).
* If mods cause sanity<=0, this is maybe not game over. Right?(had it modify health to -1 and sanity to 5[implemented])
* 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.
* Stores re-stock inventory items.
* Process events (**#10**)
* … ?
* … ?
* … ?
* eod\_mods Summary Screen  
  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) Left list: End of day mods list. Click to display the story.  
    2) Right list: 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.
    - 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)*.)
    - 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.
    - 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!
    - 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]!
    - … ?
    - … ?
    - … ?

Idea

###### #X Make hours not go negative on work.

… 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!?".

Idea

###### #X 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…
* … ?

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  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.

Programming strategy

###### #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