```
    def action_your_decision(self):
        if len(self.brain.decisions) > 0:
            action = self.brain.decisions[-1]
        else:
            action = False

        if action:
            if not self.in_dungeon: #WE ARE IN TOWN!!!
                if action == 'move_to_monster':#is this redundant? You cannae duel without finding a monster? Entry action might be useful, though ...
                    if self.move_to_monster(self.brain.contract.target_name):
                        self.brain.decisions.pop(-1)
                    else:
                        self.brain.decisions.pop(-1)
                        #monster not found ...

                elif action == 'duel': #fight to the death with the monster ...
                    if self.move_to_monster(self.brain.contract.target_name): #code allows for the possibility of monster moving, or a break in the duel process
                        for monster in monster_list:
                            if [ self.x, self.y ] == [ monster.x, monster.y ]:
                                adversary = monster
                                break
                        self.duel(adversary)
                    else:
                        self.brain.decisions.remove('duel') #should empty the brain of the last decision ...

                elif action == 'buy_cargo':
                    if libtcod.random_get_int(0,0,1) == 1: #50 / 50 whether we find somewhere else ...
                        shops = self.find_another_shop_item()
                        self.move_to_new_shop(shops[0][1]) #move to shop, by faction, of 'nearest' (or most attractive) shop.
                    self.browse_stock() #need a better function to assess whether the item is better than the one we hold?
                    self.brain.decisions.pop(-1)
                elif action == 'sell_cargo':
                    if libtcod.random_get_int(0,0,1) == 1: #50 / 50 whether we find somewhere else ...
                        shops = self.find_another_shop_item()
                        self.move_to_new_shop(shops[0][1]) #move to shop, by faction, of 'nearest' (or most attractive) shop.
                    self.sell_items()
                    self.brain.decisions.pop(-1)
                elif action == 'go_home':
                    self.go_home()
                    self.brain.decisions.remove('go_home')

                elif action == 'rest':
                    if self.hp['current'] < self.hp['max']:
                        self.rest()
                    else:
                        self.brain.decisions.remove('rest') #should empty us?

                elif action == 'move_shop':
                    if self.brain.contract.mission_type == 'buy':
                        shops = self.find_another_shop_item(self.brain.contract.target_name)
                    elif self.brain.contract.mission_type == 'sell':
                        shops = self.find_another_shop_item(self.brain.contract.target_name)
                    else:
                        shops = self.find_another_shop_social()

                    if [ self.brain.contract.mission_type, shops[0][1] ] == ['buy', 'Merchants']:
                        self.move_to_new_shop(shops[1][1]) #second 'nearest' shop because we can't buy at the merchants house
                    else:
                        self.move_to_new_shop(shops[0][1]) #move to shop, by faction, of 'nearest' (or most attractive) shop.
                    self.brain.decisions.pop(-1) #empty this action from the list of decisions
```

```
            elif action == 'socialise':

                heroes_here = []

                for hero in town_heroes:
                    if hero != self:
                        if [hero.x, hero.y] == [self.x, self.y]:
                            heroes_here.append(hero)

                if heroes_here:
                    hero_chat = pick_random_from_list(heroes_here)

                    log_or_not = libtcod.random_get_int(0,0,4)

                    if log_or_not == 1:
                        self.activitylog['history'].append(pick_random_from_list(event_chat_with_pre) +
hero_chat.name)
                    elif log_or_not == 2:
                        self.activitylog['history'].append(hero_chat.name + pick_random_from_list(
event_chat_with_post))
                    else:
                        pass
                    self.brain.decisions.pop(-1)

                else: #no one around ... lets wait for someone to come?
                    if 'dead_around_here' in self.brain.flags:
                        self.brain.decisions.append('move_shop') # fuck it lets try somewhere else
                        self.brain.flags.remove('dead_around_here')
                        self.brain.flags.remove('quiet_around_here')

                    elif 'quiet_around_here' in self.brain.flags:
                        self.brain.flags.append('dead_around_here')
                    else:
                        self.brain.flags.append('quiet_around_here')

            elif action == 'buy':
                self.browse_stock(self.brain.contract.target_name) #need a better function to assess whether the item is
better than the one we hold?
                self.brain.decisions.pop(-1)
            elif action == 'sell':
                self.sell_items(self.brain.contract.target_name)
                self.brain.decisions.pop(-1)
            elif action == 'descend':
                self.move_to_dungeon(self.brain.contract.target_name)
                self.brain.decisions.pop(-1)

            elif action == 'find_item': #shouldn't be here, but might be a holdover from some shite?
                self.brain.decisions.pop(-1) # just get rid of it
            elif action == 'resolve_contract':
                self.resolve_contract()
                self.brain.decisions.pop(-1)
            elif action == 'raise_dead':
                if 'healer' in self.brain.personality.perk:
                    for corpse in dead_heroes:
                        hero = corpse[0]
                        if [self.x, self.y] == [hero.x, hero.y]: #check if there is a hero here
                            if corpse[1] < hero_rot_time: # the corpse is still there
                                hero.hp['current'] = hero.hp['max'] #ressurect! To max points, otherwise they just die again ...
                                dead_heroes.remove(corpse)
                                town_heroes.append(corpse[0])
                                self.gain_experience(10)
                                self.activitylog['history'].append('I have restored the life force to a soul
in need.')
                                self.brain.decisions.pop(-1)
```

```
                        hero_message(str(self.name) + ' has raised ' + str(hero.name) + '!', libtcod.
light_yellow, libtcod.dark_yellow)
                        if hero.faction == 'Necromancers':
                            hero.activitylog['history'].append(self.name + ' has brought me back from
death.')
                        else:
                            hero.activitylog['history'].append(self.name + ' has given me back life!')

            elif action == 'defile_corpse':
                if 'defiler' in self.brain.personality.perk:
                    for corpse in dead_heroes:
                        hero = corpse[0]
                        if [self.x, self.y] == [hero.x, hero.y]:
                            if corpse[1] < hero_rot_time: # the corpse is still there
                                dead_heroes.remove(corpse)
                                hero_message(hero.name + ' is defiled!', libtcod.light_red, libtcod.dark_red)

                                self.activitylog['history'].append('I have gained power from the corpse of
another.')
                                self.gain_experience(10)
                                self.brain.decisions.pop(-1)

                                gender_test = libtcod.random_get_int(0,0,1) # now make a replacement ...
                                if gender_test == 0:
                                    gender = 'm'
                                else:
                                    gender = 'f'

                                birth_hero(gender)
                                hero_message(hero_list[-1].name + ' takes up the fight!', libtcod.light_yellow,
libtcod.dark_yellow)

            elif action == 'find_corpse':
                corpse_location = self.find_corpse(self.brain.contract.target_name)
                if corpse_location[2]: # hero in a dungeon
                    for n in range(corpse_location):
                        ##AAARGH WHAT ABOUT KNOWING WHICH DUNGEON TO GO TO? SHALL WE CHANGE IT
HERE?

                        self.brain.decisions.append('descend')
                    self.brain.contract.target_name = map[corpse_location[0]][corpse_location[1]].name
# reassign target name from corpse, to dungeon
                else: # hero on the overworld
                    self.move_to_map_location(corpse_location[0], corpse_location[1])
            elif self.in_dungeon:
                random_event = libtcod.random_get_int(0,0,100)
                dlev = map[self.x][self.y].dungeon.base_level
                if action == 'descend': # take on an aggressive manner
                    if random_event < 10:
                        self.dungeon_descend()
                        self.brain.decisions.remove('descend')
                    elif random_event < 30:
                        self.fight(generate_encounter( 1, dlev, 1, dlev + ( self.in_dungeon / 3 ) ) )
                    elif random_event < 95:
                        pass
                    else:
                        self.find_item()
                elif action == 'find_item': # take on an exploratory role
                    if random_event < 10:
                        self.find_item()
                    elif random_event < 25:
                        self.fight(generate_encounter( 1, dlev, 1, dlev + ( self.in_dungeon / 3 ) ) )
                    elif random_event < 95:
```

```
            pass
        else:
            self.find_item()
    elif action == 'go_home': #defensive, let's get outta here!
        if random_event < 25:
            self.leave_dungeon()
        elif random_event < 35:
            self.fight(generate_encounter( 1, dlev, 1, dlev + ( self.in_dungeon / 3 ) ) )
        else:
            pass
```