## class EventManager:

The class that will generate a random event.

def generate\_event(self, progression):

Generate a random event.

def get\_multipliers(self):

Return the multipliers of the disaster.

## class Application(tk.Tk):

The class Application is the main application of the class.

def update\_music(self):

Play the music, this can be muted.

def start\_startscreen(self):

Show the start screen.

def start\_simulation(self):

Show the simulation panel.

def load\_simulation(self):

Load the start screen from a previous simulation.

def save\_file(self, ss):

Save the current simulation.

def end\_frame(self, index, information):

Show the appropriate end screen.

def load\_file(self, frame):

Load the StarSystem object from the save file.

def enable\_save\_file(self, x):

Enable the save button.

def window\_size(self, width, height):

Sets the size of the window.

def on\_exit(self):

Is called upon closure.

## class StartPage(tk.Frame):

This class is the start window.

## class MainPage(tk.Frame):

This is the class with all other canvasses and objects.

def load\_startpage(self, ss):

Load the start page.

def research\_focus(self, index):

Enable all buttons except the current focus

def create\_canvas(self):

Create window to draw figures on.

def thread\_make(self):

create a thread for drawing the planets.

def show\_planets(self, obj, planets):

Make new PlanetDrawing objects who draw themselves.

def get\_random\_angle(self, number):

Generate number of angles who differ 20 degree with the others.

def update\_canvas(self):

Make planets move.

def auto\_update\_change(self):

Change state when button is pressed.

def create\_progressbar(self):

Create the progressbar.

def create\_message\_window(self):

Create message window. Used for communication with user.

def show\_message\_window(self, title, body\_text):

Show message window. Used for communication with user.

def create\_info\_frame\_planet(self, startinfo, main=False):

Show planet information, if MainPlanet set, only this information will be shown.

def update\_info\_frame\_label(self, startinfo, planet=None):

Update the information labels.

def set\_main\_planet(self, planet):

Remember the main planet and asks for points set.

def next\_turn(self):

Generate next turn.

def planet\_color(self):

Create color code.

def instruction\_path(self, x):

Show instructions to guide the user.

## class PlanetDrawing:

Class object who contains the planet drawing and the rotation ring.

def move\_obj(self):

set drawing on new coordinates.

def show\_planet\_info(self, event):

Get information about the planet.

def change\_color(self, color):

Change color of planetdrawing, only used for main planet.

## class MyPopupWindow:

Asks user for point distribution.

def ok\_enter(self, event):

Calls method ok when used.

def ok(self):

Checks if the values are valid.

## class EndPage(tk.Frame):

Shows the end frame.

def show\_end(self, index, information):

Show end text.

## class MainPlanet(Planet):

The class that keeps all the info about the organism and is more detailed than Planet.

def spend\_points(self, points):

Spend the technology points in the beginning of the simulation.

def set\_research\_focus(self, index):

Method to choose focus to upgrade technologies.

def update\_research\_focus(self):

Check if research is complete, and the focus needs to be chosen again.

def change\_base\_values(self, event):

Choose the type of method to call, depending on the type of event.

def update\_multipliers(self, multipliers):

After a disaster, update the multipliers.

def update\_technologies(self, tech\_index):

When a breakthrough has appeared, it updates the right technology.

def update\_variables(self):

Update all variables separately, through its correct methods.

def calc\_atmosphere(self):

Calculate the atmosphere, and regenerate the multiplier.

def calc\_temperature(self):

Calculate the temperature depending on the atmosphere and distance.

def calc\_usable\_landmass(self):

Calculate the usable landmass, depending on archeology and agriculture, and its multiplier.

def calc\_life\_quality(self):

Calculate the life quality, depending on the temperature,landmass, engineering, and population health.

def calc\_total\_population(self):

Calculate the total population, depending on life quality, and its current population.

def calc\_progression(self):

Calculate the progression, depending on the population, and the life quality, and a few technologies.

def show\_information(self):

Print the information about the planet, like distance, name of planet, and more relevant information.

def cache\_population(self):

Save the population, to calculate the difference in the next turn.

## class Planet:

Class that keeps basic info about and represents a planet.

def calc\_temperature(self):

Calculate the temperature depending on distance and atmosphere.

def planet\_quality(self):

"Calculate the planet quality, depending on the temperature, atmosphere and landmass.

def show\_information(self):

Display the information of the planet, like name, radius and more useful information.

## class StarSystem:

Main class that keeps all the info and planets.

def set\_up(self):

Setup, to read in all the file names, and give the planets different distances.

def set\_main\_planet(self,planet):

"Let the star system know, what is the main planet, and put it in the correct list.

def next\_turn(self):

Simulate the next turn, and call all the methods to update the values.

def check\_winning\_condition(self):

Check if the game is either won, or lost, depending on progression and population.