# Save and load function (implementation)

The user might want to save mid-simulation. To be able to do this, a function to save and load is implemented. The simulation can only be saved when a main planet has been chosen, and a focus has been set. To do this, the method save\_file() was created. The object StarSystem is given as parameter with the method, and that object is written into a file. This method uses the built in module Pickle. So it is saved as a “.pkl” file.

To load that file again, the method load\_file() is used. This time the frame is the parameter of the method. First, the object StarSystem is loaded from the save file. Next, the method load\_startpage() is called from the object frame to update the GUI. StarSystem is the only parameter of this method.

In load\_startpage() the StarSystem of frame is changed to the recently loaded value. Because there is another star system, all its parameters have to be updated. All the values of the progression, and the progression bar are changed and updated in the GUI. The frame needs to know that there is another main planet, and so the value of main\_planet is changed to the current main planet. create\_canvas() will be called to create the canvas with all the planets and sun. Next it in line the info window is updated with the right information, this covers the technologies, turns, etc. The last things that need to happen, to successfully load, are that the buttons need to be updated so that the right one is selected, the message window is updated, the music is enabled and at last the method update\_canvas() is called to rotate the planets.

# GUI Implementation

The GUI consists of multiple methods, which are each used for a separate purpose. The method research\_focus on its part will check if the thread for drawing is already live. Depending on if it is, the program will first ask the user, to either choose a focus, through the code in instruction\_path(), or change the focus at the moment and change the related buttons, this being the technology buttons at the top.

A rather peculiar method is thread\_make(). The method will create a separate thread for controlling and altering the canvas. The code that the thread will run is the method update\_canvas(). In this method all the planets are moved. The reason the method is made a thread, is that without it, there would be two unfortunate choices. Either sleep() is not used, and the drawing of the planets happens to fast, and rotate at the speed of light. Or either the sleep method is used, but you can almost never use the buttons on the application, because the whole code is sleep mode. So the choice was made to use a separate thread for the drawing, so the sleep could be used.

The method create\_canvas()will create the canvas, and with that settle its background, size and location. It will also call the method show\_planets(), which on its turn will place the planets on the canvas.

The method update\_canvas() will move every planet by calling the planet.move\_obj() for each planet in the list of star system, which it holds as a parameter. At last this method will then use the time.sleep(). All of this code will be put into an infinite loop, so it will loop until either the separate thread is deleted, or the application is stopped.

show\_planets() is the method that will draw the planets. The parameters of this method are a list of the planets, and the canvas. It will then draw the planets, using the provided list which generates objects of the type planet\_drawing(). The method Get\_random\_angle() will generate a list of random angles, not more than 20 degrees separated each, and as much as there are planets. These are then placed together on the canvas, depending on the angles, added with the associated movement circles.

The methods create\_progressbar, create\_message\_window, create\_info\_frame\_planet are all pretty self-explanatory. The progress bar, message window and the frame with the info of the planet are all created.

update\_info\_frame\_label will calculate the new values of each property of the planet and species. It will also show the difference with the last value, to have a feeling about how fast it differs.

A method had to be made to choose a main planet, for this the method set\_main\_planet() was created. It will use the methods thread\_make(), and create\_info\_frame\_planet(). It will also make a window pop up with the division of the technology points. The create\_info\_frame\_planet() method will use an auto scale to scale all elements to show.

next\_turn() is most likely the key method of the whole simulation, it will call all the calculations in the right order. These calculations, the formulas, and the thinking behind it are all explained in part 4.2 Implementation.

**Thread**: A thread can be seen as a smaller process inside a process There can be multiple threads inside the process, and they all have access to the same sources as the other threads inside the process.

# Event Manager implementation:

The class event, beside its constructor, only has two methods, this being get\_multipliers() and generate\_event(). The method get\_multipliers() will return a list, with all its necessary multipliers, for example the landmass\_multiplier.

The method generate\_event() only has one parameter, and that is the progression of the species. It will first calculate the chance of a disaster, depending on the progression. Then, depending on if it is a disaster, or a breakthrough, it will do different things. If it is a disaster, it will pop an event out of the disaster\_list or breakthrough\_list. After choosing an event, that has the right progression values, it will set the right values for everything.