Power Systems and Thermal Management for the UK Space Design Competition

Theo Macklin

William Laver

Version 1.0

UK Space Design Competition

August 20, 2023

Contents

1 Introduction

Power is an ever-present, if unstated, concern in the challenges at the regionals and nationals of the UK Space Design Competition (UKSDC): it forms the backbone of life-support considerations as well as being a significant limiting factor on the effectiveness of your settlement's operations. Being so critical to almost all designs, it was reasoned that we should provide a little more support for the most common choices of power source within the UKSDC. The two power sources options discussed in this report are by no means an endorsement of them, nor should they limit your imagination.

Despite this, the mathematics of heat rejection in space, are more fundamental and should be obeyed, no matter your power source. This is an often-neglected, often unstated requirement within the request for proposal. However, just because the customer has not specifically asked you to detail how you will handle this challenge, it does not mean that the problem does not exist! Just as we would expect you, without a specific prompt, to explain how you will dispose of the irreducible waste generated by your waste processing system, we expect you to dispose of your waste heat. The maths for heat dispation has been formulated in such a way to be largely independent of your choice of power source, allowing for multiple options to be easily compared during the competition.

2 Power Systems

The concept of power is one whose technical meaning is often lost in an era where it is possible to flip a switch and have things work. In this guide we will use the technical definition of power as "energy transferred per second". This leaves the equally nebulous definition of energy, which we will simply acknowledge as a "wibbly-wobbly... thing measured in Joules"!

In practice, we often consider electrical power or mechanical power (the turning of a shaft that is able to be made to do things without stalling) to be the be-all-and-end-all of our supply requirements. In practice, we must almost always convert some other kind of energy into our target form: in a solar power plant, photonic energy is absorbed into a band-gap potential which then drives an electrical potential, from which we can source an electrical current; in a wind-turbine, the wind is decelerated as it is forced to push on a blade, which applies a moment on an axle, which turns a gearbox, which turns a generator; in a fossil-fuel plant,

chemical energy is released to “high-grade heat” which boils high pressure water, which expands through a turbine, turning it and a generator.

In each of these conversions, the second law of thermodynamics demands that we lose some of our energy to random vibrations of the atoms involved – “low grade heat”. This is why a gearbox will feel hot while running, or your computer needs a fan to crunch all those ones and zeros. In some cases, the second law can be made to tell us how the minimum amount of energy we have to lose during a conversion, and engineers chase down a target, while in others cases, engineers simply seek to make a process as loss-less as possible.

2.1 Nuclear Power

Even if you already know how a nuclear power station creates electricity you should not skip this section, as it is important that you familiarise yourself with the terminology that will be used in this guide.

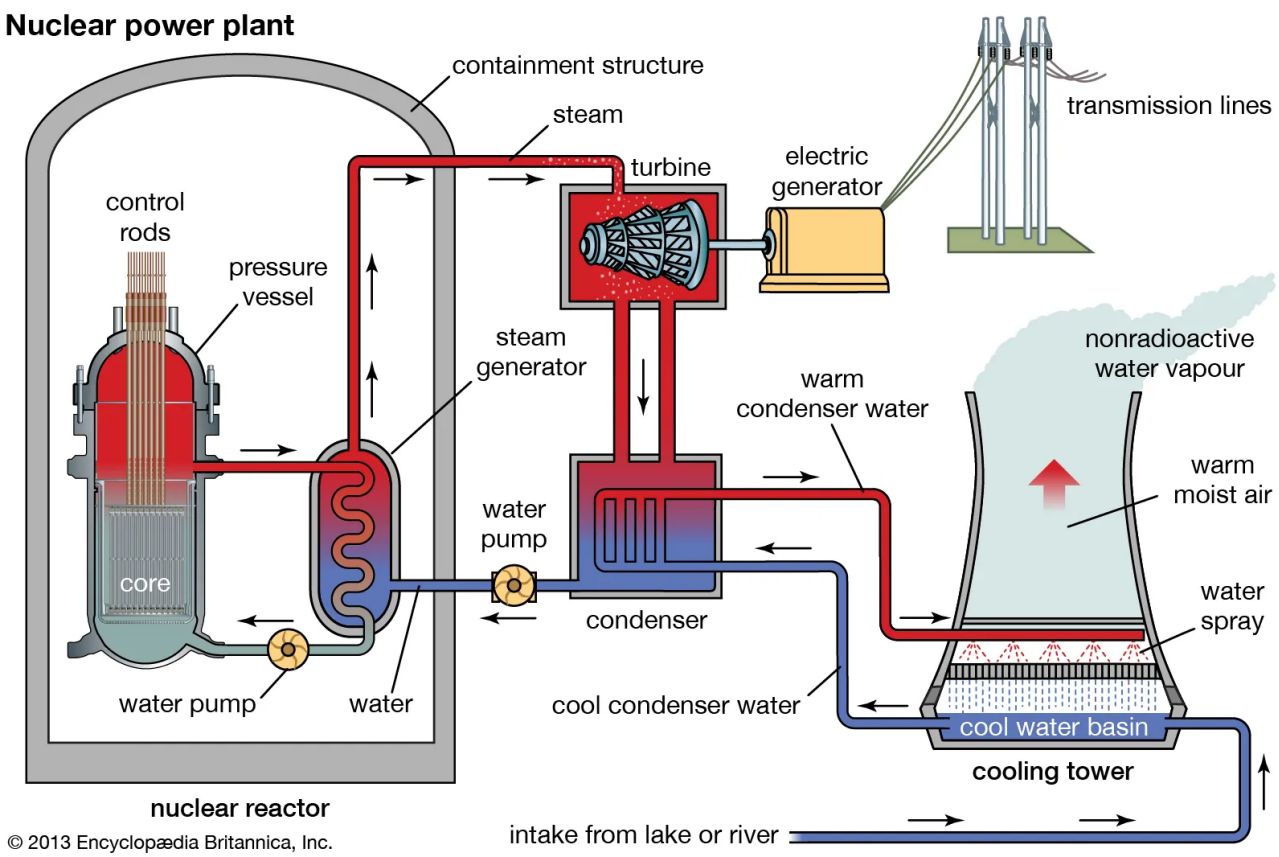

Figure 1: Nuclear power plant diagram, image from from Martin, 2022.

Figure 1 shows the basic workings of a nuclear fission power plant. Starting from the left, the nuclear fuel releases heat that is transferred to the primary fluid in the reactor (red/grey liquid), this primary fluid cools the nuclear material to keep the reactor in a safe temperature range and extracts the thermal energy. The primary fluid then transfers heat to the secondary fluid (usually water) in a heat exchanger to create steam. This high-pressure steam is then used to drive a turbine or a series of turbines, expanding and losing energy as it drives the motion of the turbine. The secondary coolant is eventually exhausted of usable thermal energy and is turned back into water in the condenser. The water, unlike the steam, is incompressible and so vastly more efficient to pump back up to a high pressure to flow back into the heat-exchanger (which is often called a *steam generator* in the nuclear industry).

Jumping back in the cycle, the condenser must have the steam's remaining heat removed by a third fluid - since heat must always travel from hot to cold. This third fluid (typically a liquid) needs a way to reject heat too: this mechanism is called the heat-sink of a power-plant and it is the lowest temperature in the whole plant. This heat-sink is usually a cooling tower or a large body of water, where the low temperature of the air, sea, or river serves to receive heat from the condenser's coolant fluid. However, none of these options are possible in space or on planets without abundant water or a dense atmosphere.

A fusion power plant, Figure 2, differs only on the heat-generation side and the possible substance of the primary coolant, which may use a liquid metal coolant (this is an option for some fission reactors, too - it's complicated on the reactor side!). In either case, the secondary coolant, the one that boils to steam, is the working fluid in a *Rankine cycle*.

Both of the nuclear subcontractors require you to quantitatively consider the cooling of the reactors that they will supply. In order to improve realism and reward effective designs, the efficiency of these reactors at generating electricity is made a function of the *Carnot Efficiency* (It's French - the "t" is silent).

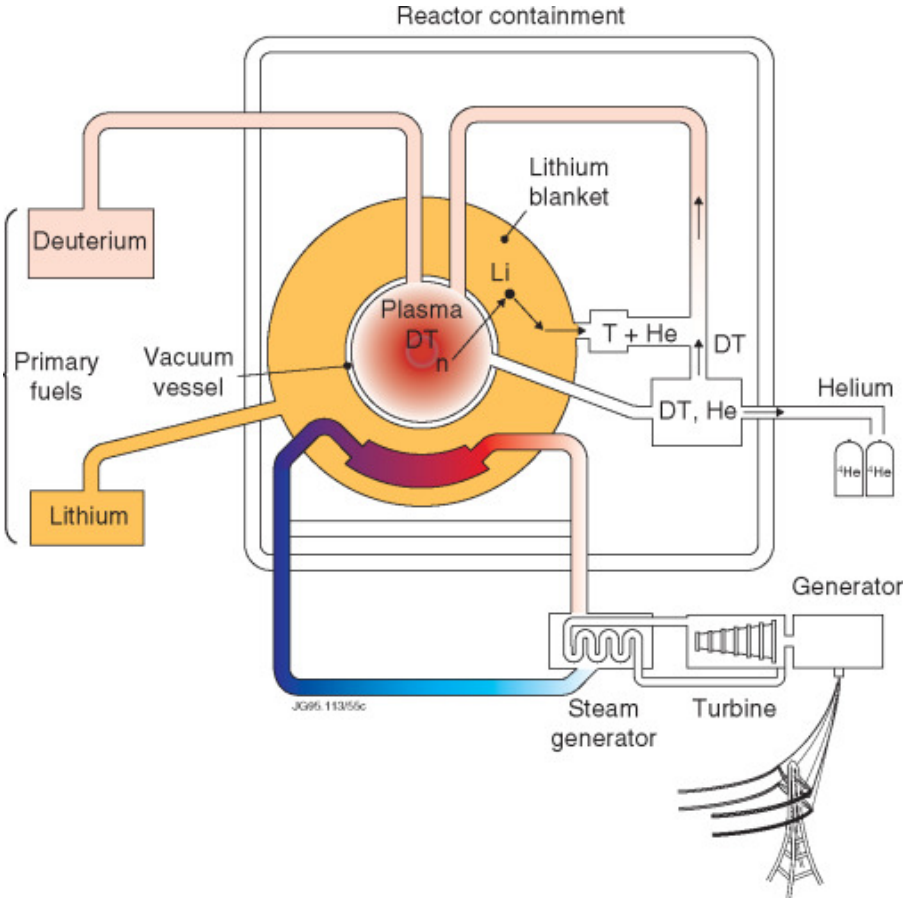

Figure 2: Speculative nuclear fusion power plant diagram, image from from McCracken and Stott, 2005. The condenser and ultimate heat-sink is neglected from this diagram.

The Carnot efficiency is a theoretical maximum amount of work that can be extracted from a heat engine (like the Rankine cycle that we're using) and is dictated by two temperatures: the temperature of the hot temperature reservoir, T_H (given by the subcontractor), and the cold temperature reservoir, T_C (dictated by how you design your heat sink). The greater the difference between these two temperatures the more work that can be theoretically extracted. The Carnot efficiency is represented by the symbol η_C, and is given by Equation 1, which should always give you a result between zero and one. You must use temperatures in kelvin.

$$\eta_C(T_H, T_C) = 1 - \frac{T_C}{T_H} \tag{1}$$

In our use, the plant (the name for the heat-source, primary coolant loop, heat exchanger, secondary-coolant plumbing, turbine, condenser, generator, etc.) that you're provided with works at 70% of the Carnot efficiency to reflect the parasitic energy losses that happen in real life. This is actually a very high proportional efficiency - many plants do not even achieve

50% of the theoretical Carnot efficiency. We shall denote this *proportional-efficiency* as η_P.

Factoring these losses in, your *thermal efficiency* (overall actual efficiency) of converting the *thermal-power* (heat output) of reactor into usable mechanical work/electricity, denoted η_T, is going to be:

$$\eta_T(T_H, T_C) = \eta_P \cdot \eta_C(T_H, T_C) \tag{2}$$

So, to work out the usable mechanical/electrical power, W, generated by a reactor and plant setup you have to take the thermal-power output of the reactor, Q (in Watts, kW, MW, or GW), and multiply it by the thermal efficiency of your system:

$$W(T_H, T_C) = \eta_T(T_H, T_C) \cdot Q \tag{3}$$

It is important to recognise that, even if we only extract a fraction of the thermal-power as useful work, since energy can neither be created nor destroyed, the rest of the thermal-power must remain as heat. In fact, this is the heat that we remove from the condenser via our heat sink, Q_W, which is a temperature marginally above that of our cold reservoir: typical rules-of-thumb suggest a temperature drop change of 10 kelvin over a heat exchanger, meaning that Q_W is extracted at 10 K above the cold reservoir temperature. The numerical value of Q_W can be calculated via Equation 4. You will notice that, in many cases, this value is greater than the useful power that we extracted!

$$Q_W(T_H, T_C) = (1 - \eta_T(T_H, T_C)) \cdot Q \tag{4}$$

Furthermore, while we have extracted useful work from our reactor, the majority of use-cases for it simply convert electricity into heat as a byproduct of its operation: e.g. computers get hot while they run, metal being cut on a lathe gets hot, rocks being smashed get hot, etc. There are a few cases where power is embedded into new forms, such as charging batteries which are then shipped away, laser beams, or the creation of high-energy materials which will not embed all of the heat associated with the power into the overall space-settlement. The heat-load for useful power is accounted for in Section 4.2.

2.2 Photovoltaic

This guide is not going to explain the ins and outs of how photovoltaic systems work, because there are plenty of guides on the internet that will do a far better job of it! Solar power is a

popular choice within the UKSDC for many reasons, chiefly revolving around its simplicity to integrate into a design and its nominally low operational overhead. While We don't deny that solar is an excellent option in many circumstances, there are significant complications that are often neglected in the UKSDC which we do consider when judging. In addition to their heat dissipation, which will be the subject of this section, we do expect you to consider solar-panels degradation, radiation damage risk, and cost. Furthermore, you should consider how these factors vary between different types of photovoltaic system.

The efficiency of a solar panel is a complicated quantity to define, as it will depend on both internal factors, such as temperature and age, and external factors, such as the angle of illumination. It is acceptable, for the UKSDC, to find a well-evidenced value for theoretical maximum efficiency, η_{lab} and to take 80% of its value to account for non-optimal illumination (e.g. a 33.0% efficient panel would be treated as 26.4% efficient). We will define this as the "nominal efficiency" of the panel, η_N. Even this large reduction is highly optimistic.

Accounting for internal factors is equally important, as these can cause even greater reductions in efficiency. Modern panels lose an increment of $\sim$0.5% efficiency per year; e.g. our η_N = 26.4% panel would be 25.9% efficient after one year of use, and only 16.4% efficient after ten years. We do not recommend reducing this value when accounting for technological improvements. The reason for our caution on this front is the violent radiation environment of outer-space (and on any planet in the solar system without an atmosphere). Indeed, the author would recommend scaling this value up for locations closer to the sun than Earth: Venusian orbit should use a value of 1.0% per year, and the orbit of Mercury a value of 3.4% per year. As a general rule for sites closer to the Sun than Earth, divide the value at Earth by the square of the distance to the Sun in astronomical units. While this seems harsh, it is physically informed, and you will get a lot more power per square meter of panel as you get closer to the Sun. The negative, per-unit form of the degradation rate with age will be written as c_a (i.e. c_a at mercury would be -0.034 per year).

The second internal factor that we would like you to consider in the UKSDC is temperature. The efficiency change can be approximated as linear with respect to the reference temperature for η_N, which is usually 25°C. Below this temperature efficiency increases and above it, reduces. There will be a limit on the efficiency increase, which you should limit to the addition of 5%, but there is no limit on the degradation. Modern solar panels swing between -0.25 and -0.43% per kelvin per above nominal (Svarc, 2022).

It is feasible that this temperature dependency could be reduced in future, potentially to a value of -0.2% per kelvin for all photovoltaic cell types. The negative, per-unit thermal sensitivity will be denoted c_T (i.e. a nominal value of -0.002). To continue the example, should our ten year old solar panel be raised to a temperature ten degrees above its reference, its efficiency would further reduce to 14.4%. It should also be noted that solar panels also have a limit on their operational temperatures: for the UKSDC it is reasonable to assume that this range is from -100 °C to +150 °C which is slightly more optimistic than in real life. Within the world of UKSDC, you should assume that a panel is completely destroyed if it goes beyond its operating temperature range, or its efficiency hits zero at any time.

Summing all these factors gives the following expression for the overall panel efficiency, η_o for a panel of age t years, an operating temperature, T_C, and a reference temperature T_{ref}. The reason for the subscript on T_C will become clear in Section 4.1.

$$\eta_o(T_C) = 0.8\eta_{lab} + c_a t + c_T \cdot (T_C - T_{ref}) \tag{5}$$

If you can justify to your CEO reasons that the prescribed values might be different, they will happily invite you to discuss your ideas with one of the judges! This efficiency means that the power output, W from your panels will be given by Equation 6, in which q_s is the power from the sun in W/m^2, and A_S^{proj} is the "projected area" of the solar panel with respect to the sun. The solar intensity, q_S, can be found via the inverse square law or found online for the distance from the sun that is of interest. Equation 7 gives a simple calculation for the projection of an area, A_S, whose *surface normal* is at an angle, α, to the direction of light from the Sun. Figure 3 similarly depicts this concept for a 3D object: a surface turned at a right angle to the sun will no longer be illuminated by it.

$$W(T_C) = \eta_o(T_C) \cdot q_S \cdot A_p^{proj} \tag{6}$$

$$A_S^{proj} = A_S \cdot \cos\alpha \tag{7}$$

The light that strikes the surface of the panel and that is not converted into useful work will either be reflected or converted into heat within the panel. The fraction that is reflected contributes quite significantly to the efficiency of the panel in most reported values, and so reflection must be accounted for when estimating the heat-load. It is reasonable, if optimistic, to assume that 5% of the unconverted energy is reflected away for the high-efficiency panels favoured in the competition, and so Equation 8 gives the waste heat, Q_W within the

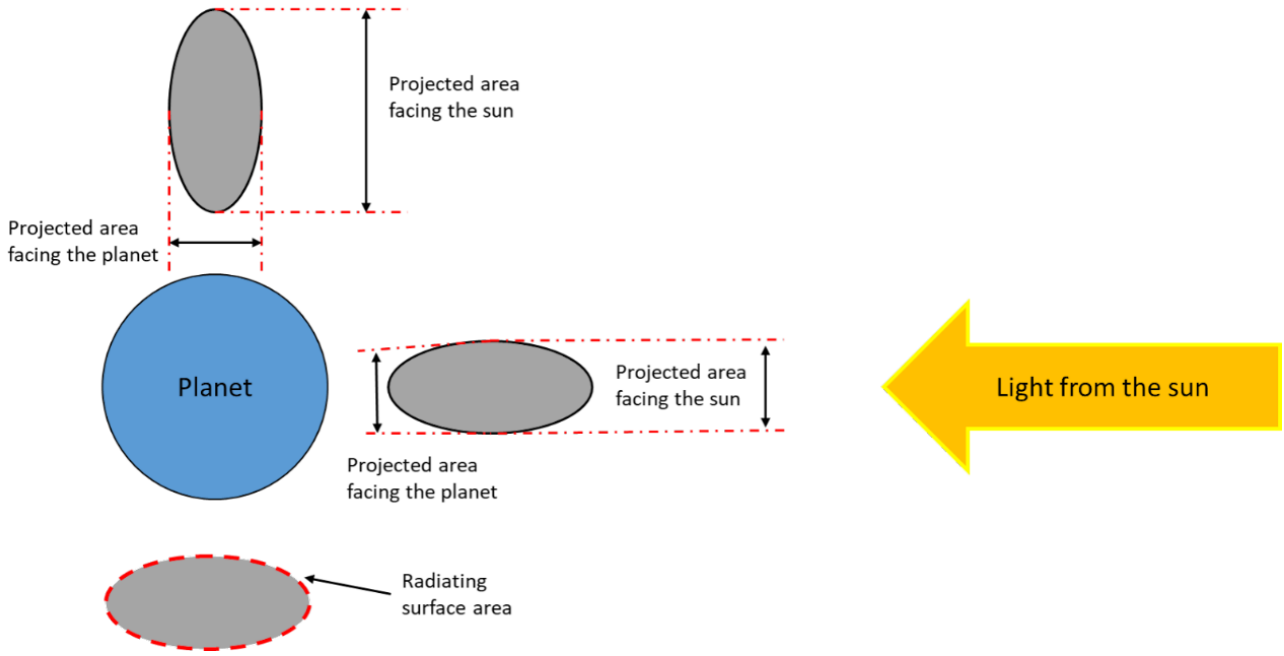

Figure 3: Depicted of the illumination of an object in different positions relative to a planet and to the Sun to demonstrate the concepts of projected areas.

panels.

$$Q_W(T_C) = 0.95 \cdot (1 - \eta_o(T_C)) \cdot q_s \cdot A_p^{proj} \tag{8}$$

Similarly to the nuclear power option, the useful energy will also be turned into heat upon its use. This is handled in Section 4.2.

2.3 Solar Thermal

Solar thermal power systems convert the heat from the sun into power via the smae kind of Rankine cycle has a nuclear reactor. These systems have the advantage of not requiring expensive semiconductors and they are less vulnerable to degradation over time.

Solar thermal systems have an absorption surface which faces the sun to collect heat: this acts as the hot reservoir at temperature T_H. T_H must be calculated via energy balance methods described in Section 3.6 from the direct solar heating described in Section 3.2. In the course of the energy balance, the heat rate that you wish to pull from the absorption surface must be specified.

Solar thermal systems also feature a radiator, which like a nuclear system, is shaded and emits waste heat out into space. The mathematics for the operation of a solar-thermal

plant are, beyond the need to specify Q in order to then calculate T_H, the same as nuclear plants.

3 Space as a Heat Sink

While the UKSDC considers more than just space stations, the dissipation of heat on a planetary surface or attached to an asteroid pose unique modelling requirements based on location, whereas free-space is generic enough to be summarised here. We have discussed how power generation generates heat, and now we have to discuss how we'll get rid of it. If we don't get rid of it, that thermal energy will accumulate onboard and make everyone quite uncomfortable quite quickly.

3.1 Emitted Heat

To dissipate heat from a spacecraft you're going to need radiators, these radiators are heated using fluid and then emit radiation in the infrared band, with a power emitted, Q_E, dependent on the temperature, T, material's emissivity, ε, and the emitting surface area, A.

$$Q_E(\varepsilon, T, A) = \varepsilon \sigma A T^4 \tag{9}$$

σ is the Stefan-Boltzman constant, which equals 5.669 $\times 10^{-8}$. This equation makes the assumption that the radiators will be "isothermal" - possessing the same surface temperature everywhere on their surface. It is also important to remember that both sides of a surface will radiate heat, so a flat plate contributes double the area of a single face.

3.2 Direct Solar Thermal Heat Received

Unfortunately, objects in space, like radiators, absorb radiation as well as emit it. This must be accounted for when calculating the heat dissipation from your radiators and the biggest contributor to this heat-load is from the sun striking the surface. This is given by Equation 10, in which a_S is a property known as the *solar absorptance* of the emitting surface, A_S^{proj} is the sun-facing surface, equivalent to solar-panel area, and q_S is as previously defined.

$$Q_S = q_S \cdot a_S \cdot A_S^{proj} \tag{10}$$

The surface area in this case is single-sided, as solar illumination will only be incident on a single side of a flat plate. In the case of solar-panel systems, this heat generation term is equivalent to Q_W, and so should be ignored to prevent double-counting of the solar heat load. This is expanded upon and written in context in Section 4.

3.3 Indirect Solar Thermal Heat Received

You may also want to consider the solar radiation which is reflected off the surface of a nearby planet back at your radiating surface. This heat contribution, Q_a, is given by Equation 11, in which a, A_p^{proj}, ϕ, R_P, and $R_{s/c}$ are the planetary *albedo*, projected area of the surface relative to the planet, the angle between the planet-Sun line and the planet-spacecraft line, the radius of the planet, and the distance from the planet to the spacecraft respectively.

$$Q_a = a \cdot q_S \cdot a_S \cdot A_p^{proj} \cdot \cos\phi \cdot \left(\frac{R_P}{R_{s/c}}\right)^2 \tag{11}$$

Keep in mind that this formula is only non zero when the spacecraft is on the light side of the planet, on the dark side there is no solar radiation to reflect. Calculation of this component is an added bonus and typically unnecessary at Regionals level, but is expected at Nationals for at least the closest planet to the settlement.

3.4 Planetary Thermal

You can also consider the amount of a nearby planet's thermal emission that is captured, Q_P, using Equation 12, where ε is the emissivity of the receiving surface and q_P is the thermal power incident on the sun-facing side of the planet divided by its total area.

$$Q_P = \varepsilon \cdot q_P A_P^{Proj} \cdot \left(\frac{R_P}{R_{s/c}}\right)^2 \tag{12}$$

This formula is an adequate approximation for when the spacecraft is on the day-side or night-side of the planet. The use of ε in this equation is a reasonable simplification: in actuality, the thermal absoptivity can be different from the emissitity, but for a surface that has reached a steady-state, they become the same. This saves you from having to research yet another coefficient!

3.5 Net Heat Rate

Bringing all these terms together, we get that:

$$Q_{net} = Q_E(\varepsilon, T, A) - Q_S - Q_a - Q_P \tag{13}$$

Where Q_{net} represents the total heat emitted from the surface. As each of the component terms are positive, if Q_{net} is positive the surface is emitting heat out into space. If Q_{net} is negative, then the surface is receiving heat. This sign-convention is actually reversed from that used in engineering: in engineering we always say that heat that comes into a system is positive and heat leaving a system is negative, but the other way around is more convenient for this application!

If we wish to consider more radiators, stars, and planets/bodies which contribute to our heat load, we can include them in our sum:

$$Q_{net} = \sum_{r}^{\text{Radiators}} Q_E(\varepsilon_r, T_r, A_r) - \sum_{s}^{\text{Stars}} Q_{S,s} - \sum_{b}^{\text{Bodies}} (Q_{a,b} + Q_{P,b}) \tag{14}$$

This is worth doing if designing a spacecraft that will operate within the Earth-Moon system. To track all of the heat-load contributions, it is recommended that you make a spreadsheet or a Python program.

3.6 Thermal Equilibrium

If there is no active heat-generation on-board the spacecraft, we will reach a passive, "equilibrium state", where the energy received is balanced by the energy emitted. This is the case where Q_{net} = 0 and so the equilibrium temperature of the emitting surface, for fixed values of ε and A, is that which gives $Q_E(\varepsilon, T, A) = Q_S + Q_a + Q_P$ (assuming a single radiator and body).

4 Radiator Design

In order to design your power system and its required cooling system, we recommend that you research and collect the relevant information about the settlement's location. This data would include the solar intensity, the albedo of any nearby planets, the radius of that planet and your distance from it, and if possible the thermal emission of that planet.

An effective way to work is to negotiate space for scalable radiators within the structure and to decide how many of them you want. You should seek to minimise the projected area of the radiators facing the Sun and, as a secondary consideration, any nearby planets. You should also research and choose a coating/material for your radiators to maximise emittance and minimise absorptance.

If you are using a heat-based power system (nuclear), you should select which reactors you think would be appropriate for your settlement and make a note of their hot reservoir temperatures. Collate all of these pieces of information in Excel or Python and implement the heat rate equations from the previous section. This rest of section will discuss how these equations should be combined to assist in radiator design.

4.1 Primary Heat Load

The equations that we have presented must be handled slightly differently in the cases of nuclear solar power systems. The reason for this is that the heat-load for solar panels comes only from the unutilised solar illumination, which is an additional heat load in the case of nuclear power. In both cases, it is important to note that we will handle the heat-load from the useful power that we have extracted, W, with a separate calculation.

4.1.1 Nuclear

For a nuclear system, heat is released from nuclear reactions in addition to the heat absorbed in its radiators. This means that the Q_{net} from the panels must be equal to this additional heat rate, Q_W, otherwise the total amount of energy on the spaceraft would increase over time - i.e, the heat we're emitting overall is equal to the waste heat from the reactor system. For a single radiator and set of additional sources, this gives Equation 15, or its rearranged form, Equation 16.

$$Q_W(T_C) = Q_E(\varepsilon, T_C, A) - Q_S - Q_a - Q_P \tag{15}$$

$$0 = Q_E(\varepsilon, T_C, A) - Q_W(T_C) - Q_S - Q_a - Q_P \tag{16}$$

Equation 16 is a form of the power-balance equation that has been made suitable for *root-finding* methods to be employed - where an appropriate value of T_C must be found to make the left-hand-side equal to zero. If you have conducted your analysis in Excel, you could make T_C an initial parameter and have a cell output the value of the the left-hand-side of

the Equation 16. You can then manually iterate until the left-hand-side is made as close to zero as possible. Alternatively, you could make use of the "Goal Seek" function in Excel (in *Data*>*What-If Analysis*>*Goal Seek*) to automatically find the minimising value of T_C. If you have conducted your analysis in Python, *SciPy*'s root-finding functions can take in a function of T_C with references to global values of ε and A and return the balancing value of T_C.

4.1.2 Photovoltaic

As solar illumination is effectively converted directly into electricity in a solar cell, the only heat "generated" as a by-product of the system is the solar energy that is absorbed uselessly by the panel. Factoring this behaviour into a model to determine the amount of power that will be generated is actually easier than for a nuclear system. The direct solar thermal heat received term, Q_S, must be removed from Equation 13, as it is now excessive (because some of that energy is converted to electricity). This term is replaced by the Q_W for solar systems, Equation 8, which is our model for solar heat absorption. There is no additional heat to dissipate, and so the root-finding form of the equation natural emerges as Equation 17.

$$0 = Q_E(\varepsilon, T_C, A) - Q_W(T_C) - Q_a - Q_P \tag{17}$$

In this case, you should pay careful attention to the panel areas in each term, as some will absorb or emit heat from both sides, and some only from one side.

4.1.3 Solar Thermal

Solar thermal power becomes more difficult to keep track of as you will have radiators that act to absorb heat and to radiate it away. For the absorbing panel, assign Q_{net} in Equation 13 or 14 to be the thermal power rate that you would receive from a reactor (Q), but as a negative because the surface is receiving power. E.g. if you want to pull 50 MW of heat into your system, set Q_{net} to - 50$\times$10^6. Converge this system to find the equilibrium temperature of the surface, which is governed by the emissivity. This temperature will act as the T_H for Equations 1, 2, 3, and 4.

It is doubly important to note that the waste-heat from the power system will be emitted at T_C, which is the temperature of the cold-side radiator, which should be in the shadow of the station. We can follow the same procedure as for the nuclear reactor's radiators for the cold side of the system: converging to find the cold temperature, acknowledging the dependence of Q_W on the value of T_C.

4.2 Atmospheric Loop Heat Load

We have discussed already that even our useful power must eventually become heat, so there is the obvious question of why we have not simply factored it into the emission demand of our primary heat load radiators. The reason for this is that these loads typically dump their heat into the atmosphere within the settlement by the nature of their use in settlement operations. This means that the dissipation of W Watts, kW, MW, or GW of heat is dumped into to the human environment. The atmosphere will, in turn, then use an external radiator as its heat sink. Given that we have to keep the atmosphere within the settlement at a comfortable temperature, we do not want to bundle the reactor heat-source and atmospheric heat-source together and get a radiator at, e.g., 60 degrees Celsius, which would imply an even hotter atmospheric temperature, which some saunas would envy!

Within the UKSDC, we call the use of power and the control of its heat load the "operational-loop", with the loop that features the habitable atmosphere being called the "atmospheric loop". The heat load within the atmospheric loop comes from more than just the useful work from our power system, W, however: humans also produce a lot of heat, Q_H.

If all consumable calories are produced on-board under power from the reactor or solar power system, and a completely closed-loop waste processing cycle is in place, the human heat source, Q_H, can be ignored since all of their energy is ultimately coming out of the useful power. An assumption here is that a portion of W has been assigned to produce foodstuffs, with that portion being equal to the total energy investment necessary to power all the humans!

If some waste is not recycled and is stored on-board or shipped out, the energy embedded in this waste can be subtracted from Q_H as it is not being released as heat. This is very difficult to calculate but for biological waste, it is approximately equivalent to its enthalpy of combustion. In most cases, it is not worth factoring this out, unless it is significant.

If food or combustibles are imported to the spacecraft/settlement, the energy they contribute to the system must be accounted for: assume that an average human, fed entirely by imported materials, will contribute 109 W of heat. This value does not reduce with reduction in felt-gravity. If diets are partially composed of imported food-stuffs, scale these loads by the fraction of imported goods in the diet.

If, as mentioned previously, power is beamed off the settlement, or exported in batteries or

energetic materials, this power or average power (for intermittent exports of stored energy), W_R, can be subtracted from W when designing the heat-sink. It is critical to acknowledge that beamed power systems are very inefficient, and so for W_R to be, for exmaple, lasered away, a much larger part of the energy budget would have to be allocated to it.

With these values calculated, no convergence is necessary, instead find solving the following for $T_{C,O}$, the emission temperature of the operational-loop heat sink, remembering to recalculate the values of Q_S, Q_a, and Q_P for the dimensions and materials of this heat-sink:

$$Q_E(\varepsilon, T_{C,O}, A) = W(T_C) + Q_H + Q_S + Q_a + Q_P - W_R \tag{18}$$

4.3 A Word on Heat Pumps

Given that we typically favour a cooler atmospheric temperature which might imply an enormous radiator, there is a viable option to make use of heat pumps to raise the "quality" (temperature) of the heat at the expense of inputting useful work.

A heat pump will transfer heat-power equal to Q_{Moved}:

$$Q_{Moved} = \eta_P W_{in}\, COP_{ideal} \tag{19}$$

Where $COP_{ideal} = \frac{T_U}{T_U - T_L}$, maxing out at about 10, η_P is the proportional efficiency of the device (you may assume a high value of 0.70), and T_U and T_L are the upper and lower temperatures within the device. COP stands for "coefficient of performance", which is used since this device takes in power and generates heat, rather than the other way around.

In practice, what this means is that Q_{moved} of heat-power is transferred from a fluid at a temperature of T_L to an upper temperature of T_U by a power input of W_{in}. The heat-pump will dissipate W_{in} at a temperature of T_U. As heat must conduct into the pump, T_L must be taken as at least ten degrees cooler than the atmospheric temperature and T_U must be at least ten degrees higher than the temperature of the heat sink that the heat-pump is feeding, T_C. For the operational loop, if you divide your heat load $(W(T_C) - W_R + Q_H)$ by $\eta_P\, COP_{ideal}$, you can find the work that you'll have to budget for the heat-pumps.

As the heat pump will convert its W_{in} to heat at T_H, it does not have to pump its own power. Assuming that W_{in} is being drawn from W, the following expression for the power consumption of the heat pumps can be found (assuming that the heat pump is being used on the

atmospheric loop.

$$W_{in} = \frac{W(T_C) - W_R + Q_H}{1 + \eta_P \, COP_{ideal}} \tag{20}$$

This allows you to select any value of T_C for the atmospheric-loop heat sink (or any operational-loop heat sink), rather than just those at just-below comfortable temperatures. Be aware that large temperature changes will dramatically reduce the COP and the heat-pump load will dominate your power budget. You may need to iterate to ensure that you can power both your settlement and your heat pumps.

It is also possible to use heat-pumps on your reactor heat sink: this may appear to allow you to extract more power, but I'm afraid this will make you worse off overall as any extra power you've generated will be more than consumed by the heat pumps!

4.4 Additional Loops

If it would be impractical for a specific system which converts a large part of W to dump its heat into the atmosphere (e.g. mineral processing), or if a system prefers to operate at a high temperature, it is viable and recommended for your settlement to include additional operational heat loops which not coupled to the atmospheric temperature.

If you have multiple, separate power systems (e.g. multiple nuclear reactors), it may be worthwhile to give each its own primary loop in order to ensure that if one goes down, it does not impact the efficiency of the other (although you may want this).

if you choose to do this, you can include additional absorption surfaces or radiators and treat each system individually using the equations in the previous sections. Remember to separate out the power/heat contributions that are going into each radiator!

5 Conclusion

We are very aware that this document is quite heavy and it is important to reiterate that this level of detail is only expected at the national finals. Thermodynamics is the most fundamental but often neglected or misunderstood aspect of space design, so we hope that this document provides some clarity on how these calculations can be approximated. We have entirely neglected discussion of how heat is transported internally, e.g. how atmospheric processing gets its heat to the radiator, as it would make this guide even longer: we leave that

problem for your research and imaginations! If you have difficulties with any of the aspects of this document, we encourage you to drop us an email or, on the day of a competition, speak to a technical advisor.

References

Martin, W. (2022). *Nuclear Power*. Encyclopedia Britannica. URL: https://www.britannica.com/technology/nuclear-power.

McCracken, G. & Stott, P. (2005). Chapter 1 - What Is Nuclear Fusion? *Fusion*. Burlington, Academic Press, pp. 1–5.

Svarc, J. (2022). *Most Efficient Solar Panels 2022*. Clean Energy Reviews. URL: https://www.cleanenergyreviews.info/blog/most-efficient-solar-panels.

A ThermalSystem.py

The following code is able to apply many of the methods discussed in this report and has been designed was a (relatively) user-friendly interface. it requires Python 3.10.0 or better to run. An example set of thermal loops are given in the if __name__... block (in which you can safely write code to use the functions in the file). As copying the code from this document might be a touch tricky, there is an uploaded script in the same location as this document: this script has been uploaded as a .txt rather than a .py for safety. If you convert this file extension, you can run it normally. Do double check that the code is the same as in this document before changing the file extension!

```
from numpy import cos, deg2rad
from scipy.optimize import newton, toms748

class Location:
    def __init__(self, distance_from_sun_au) -> None:
        self.distance_from_sun_au = distance_from_sun_au
        self.q_S = 1380.73 / (distance_from_sun_au**2)


class Surface:
    stefan_boltzmann_constant = 5.670374419e-8

    def __init__(
        self,
        emission_area,
        sun_facing_area,
        emissivity          = 0.5,
        absorptance         = 0.5,
        angle_to_sun_deg    = 0,
        temperature_offset  = 0,
    ) -> None:
        """
        emission_area is the radiative area. This is often both sides of a plate.\n
        sun_facing_area is the area that solar heating will act on. This is only ever one side
             of a plate.\n
        emissivity is what is says it is.\n
        absorptance is the solar absorptance.\n
        angle_to_sun_deg is the angle of the surface to the sun. An angle of 0 degrees means
            that the normal to the surface is pointing directly at the sun. An angle of 90
            degrees means that the plate is edge on and not illuminated.\n
        temperature_offset is an offset in the isothermal temperature of the panel relative to
             the perceived temperature of the system. This is useful for representing active
            cooling of solar panels.
        """
```

```
        self.emissivity          = emissivity
        self.absorptance         = absorptance
        self.emission_area       = emission_area
        self.sun_facing_area     = sun_facing_area
        self.sun_facing_area    *= cos(deg2rad(angle_to_sun_deg)) if angle_to_sun_deg else 1
        self.temperature_offset = temperature_offset
        self.T                   = None

    def set_T(self, T):
        self.T = T + self.temperature_offset

    ##--Q_E
    def radiated_heat(self, T):
        T_perceived = T + self.temperature_offset
        return self.emissivity * self.stefan_boltzmann_constant * self.emission_area * (
            T_perceived**4)

    ##--Q_S
    def direct_solar_thermal_heat_received(self, loc : Location):
        return loc.q_S * self.absorptance * self.sun_facing_area

    def __repr__(self):
        return f"Temperature = {self.T:.1f} K"


def clean_surfaces_input(surfaces : Surface | list[Surface] | dict[str : Surface]):
    if isinstance(surfaces, Surface):
        surfaces = [surfaces]
    elif isinstance(surfaces, dict):
        surfaces = surfaces.values()
    elif isinstance(surfaces, list):
        pass
    else:
        raise ValueError("Invalid format for 'surfaces' or 'radiators': must be a Surface, 
            list of Surfaces, or a dict of Surfaces. Quitting.")
    return surfaces


class HeatSource:
    def __init__(self, q_w = 0) -> None:
        self._q_w = q_w
        self.T_C = None

    def q_w(self, *args):
        return self._q_w

    def __repr__(self):
        return f"Cold-Side Temperature = {self.T_C:.1f} K\nWaste Heat = {self.q_w():.1f} W"

class PowerSource(HeatSource):
    def __init__(self, w = None, q_w = None) -> None:
```

```
        self._w = w
        super().__init__(q_w)

    def w(self, *args):
        return self._w

    def __repr__(self):
        return super().__repr__() + f"\nUseful Power = {self.w():.1f} W"


class Turbine(PowerSource):
    eta_p = 0.70

    def __init__(self, qt, T_H) -> None:
        self.qt     = qt
        self.T_H    = T_H

        super().__init__(0,0)

    def eta_C(self, T_C):
        return 1 - T_C / self.T_H if T_C < self.T_H else 0.0

    def eta_T(self, T_C):
        return self.eta_p * self.eta_C(T_C)

    def w(self, T_C = None):
        if T_C == None:
            T_C = self.T_C
        return self.qt * self.eta_T(T_C)

    def q_w(self, T_C = None):
        if T_C == None:
            T_C = self.T_C
        return self.qt - self.w(T_C)

    def __repr__(self):
        return super().__repr__() + f"\nHot-Side Temperature = {self.T_H:.1f} K"


class Reactor(Turbine):
    reactors = {
        "fission_frontiers" : {
            "tarasque"  : {"qt" : 1e9,   "T_H" : 1300},
            "guivre"    : {"qt" : 2e9,   "T_H" : 1100},
            "peluda"    : {"qt" : 1e9,   "T_H" : 600},
            "lindworm"  : {"qt" : 500e6, "T_H" : 950},
            "wyvern"    : {"qt" : 250e6, "T_H" : 600},
        },
        "fusion_founders" : {
            "standard_reactor" : {"qt" : 300e6, "T_H" : 1300},
        }
```

```
    }

    def __init__(
        self,
        qt = None,
        T_H = None,
        brand = None,
        model = None,
    ) -> None:
        """
        qt is the thermal power in Watts if you are entering a custom reactor specification.\n
        T_H is the hot reservoir temperature in Kelvin if you are entering a custom reactor
            specification.\n
        brand is the name of the reactor manufacturer for a standard installation. Either "
            fission_frontiers" or "fusion_founders".\n
        model is the name of the reactor model for a standard installation. Fission Frontiers
            offer: "tarasque", "guivre", "peluda", "lindworm", "wyvern", while Fusion Founders
             offer: "standard_reactor" only.
        """
        custom  = qt and T_H
        library = brand and model
        if not custom and not library:
            raise ValueError("Reactor␣is␣not␣adequately␣defined:␣both␣of␣'qt'␣and␣'T_H'␣or␣
                both␣of␣'brand'␣and␣'model'␣must␣be␣complete!␣Quitting.")
        elif custom and library:
            raise ValueError("Reactor␣is␣over␣defined:␣both␣of␣'qt'␣and␣'T_H'␣or␣both␣of␣'
                brand'␣and␣'model'␣must␣be␣complete,␣not␣all␣of␣them!␣Quitting.")
        elif custom:
            super().__init__(qt, T_H)
        elif library:
            try:
                reactor_data = self.reactors[brand.lower()][model.lower()]
                super().__init__(reactor_data["qt"], reactor_data["T_H"])
            except:
                lines = [
                    f"Reactor␣'{model}'␣by␣'{brand}'␣cannot␣be␣found!␣The␣valid␣brands␣and␣
                        their␣models␣are:"
                ]
                for brand in self.reactors.keys():
                    lines.append(f"\t{brand}:")
                    for model in self.reactors[brand].keys():
                        lines.append(f"\t\t{model}")
                raise ValueError("\n".join(lines))
        else:
            raise ValueError()


class SolarAbsorber(Turbine):
    def __init__(
        self,
        loc,
```

```
        qt : float | int | list[float] | list[int] | dict[str : float] | dict[str : int] = [],
        surfaces : Surface | list[Surface] | dict[str : Surface] = [],
    ) -> None:
        sum_qt = sum( qt if isinstance(qt, list) else [qt] )

        if isinstance(qt, float) or isinstance(qt, int):
            sum_qt = qt
        elif isinstance(qt, dict):
            sum_qt = sum( [ v for v in qt.values()] )
        elif isinstance(qt, list):
            sum_qt = sum( qt )
        else:
            raise ValueError("Invalid␣format␣for␣'surfaces':␣must␣be␣a␣Surface,␣list␣of␣
                Surfaces,␣or␣a␣dict␣of␣Surfaces.␣Quitting.")

        self.surfaces = clean_surfaces_input(surfaces)

        sum_qs = sum( [surf.direct_solar_thermal_heat_received(loc) for surf in self.surfaces]
             )

        if sum_qt > sum_qs:
            raise ValueError(f"The␣sum␣of␣qt␣(~{sum_qt:.0f}␣W),␣the␣power␣pulled␣from␣the␣
                solar␣absorber,␣is␣greater␣than␣the␣energy␣that␣it␣is␣receiving␣(~{sum_qs:.0f
                }).␣Quitting.")

        def absorber_balance(T_H):
            nonlocal loc, qt, self
            Q_W = sum_qt
            Q_S = sum_qs
            Q_E = sum( [surf.radiated_heat(T_H) for surf in self.surfaces] )
            return Q_E - Q_W - Q_S

        T_H = newton(absorber_balance, 273.15, tol = 1e-3, maxiter = 2000)
        self.q_incident = sum( [surf.direct_solar_thermal_heat_received(loc) for surf in self.
            surfaces] )
        self.q_emitted  = sum( [surf.radiated_heat(T_H) for surf in self.surfaces] )
        for surf in self.surfaces:
            surf.T = T_H

        super().__init__(sum_qt, T_H)


class Photovoltaic(PowerSource):
    non_optimal_illumination_factor = 0.8
    c_a_at_earth                    = -3.4e-2
    c_t                             = -0.2e-2
    reflection_fraction             = 5e-3
    temperature_limits              = {"min" : -100 + 273.15, "max" : 150 + 273.15}

    def __init__(
        self,
```

```
    loc,
    eta_lab,
    T_ref,
    age_years,
    surfaces : Surface | list[Surface] | dict[str : Surface] = [],
) -> None:

    self.loc         = loc
    self.eta_lab     = eta_lab
    self.T_ref       = T_ref
    self.age_years   = age_years
    self.surfaces    = clean_surfaces_input(surfaces)

    for surf in self.surfaces:
        if surf.absorptance != 0:
            print("Setting␣absorptance␣of␣surface␣assigned␣to␣photovoltaic␣power␣source␣to
                ␣zero␣to␣prevent␣double␣counting␣of␣heat␣load.␣Remember␣to␣include␣this␣
                surface␣as␣a␣radiator␣in␣the␣ThermalLoop!")
            surf.absorptance = 0

    self.total_collection_area  = sum( [surf.sun_facing_area for surf in self.surfaces] )
    self.total_radiative_area   = sum( [surf.emission_area for surf in self.surfaces] )
    super().__init__()


def eta_N(self):
    return self.non_optimal_illumination_factor * self.eta_lab

def c_a(self):
    return self.c_a_at_earth / (self.loc.distance_from_sun_au**2)

def eta_overall(self, T_C):
    if T_C <= self.temperature_limits["min"]:
        return 0.0
    elif T_C >= self.temperature_limits["max"]:
        return 0.0
    if not "eta_non_thermal" in self.__dict__:
        self.eta_non_thermal = self.eta_N() + self.age_years * self.c_a()
        self.temperature_limits["max"] = (- self.eta_non_thermal / self.c_t) + self.T_ref
        print(f"Panel␣max␣temperature␣=␣{self.temperature_limits['max']:.1f}␣K␣({self.
            temperature_limits['max']␣-␣273.15:.1f}␣degC)")
    return self.eta_non_thermal + self.c_t * (T_C - self.T_ref)

def w(self, T_C = None):
    if T_C == None:
        T_C = self.T_C
    return self.eta_overall(T_C) * self.loc.q_S * self.total_collection_area

def q_w(self, T_C = None):
    if T_C == None:
        T_C = self.T_C
```

```
        return (1 - self.reflection_fraction) * (1 - self.eta_overall(T_C)) * self.loc.q_S *
            self.total_collection_area

    def __repr__(self):
        return "\n\n".join([str(surf) for surf in self.surfaces]) + f"\nUseful Power = {self.w
            ():.1f} W\nWaste Heat = {self.q_w():.1f} W"


class ThermalLoop:
    def __init__(
        self,
        loc : Location,
        heat_sources : HeatSource | list[HeatSource] |dict[str : HeatSource] = {},
        radiators : Surface | list[Surface] | dict[str : Surface] = [],
    ) -> None:
        self.loc     = loc
        self._T_C    = None

        if isinstance(heat_sources, HeatSource):
            self.heat_sources = [heat_sources]
        elif isinstance(heat_sources, dict):
            self.heat_sources = heat_sources.values()
        elif isinstance(heat_sources, list):
            self.heat_sources = heat_sources
        else:
            raise ValueError("Invalid format for 'heat_sources': must be a HeatSource, list of
                HeatSources, or a dict of HeatSources. Quitting.")

        self.radiators = clean_surfaces_input(radiators)

        hsT_H = [hs.T_H for hs in self.heat_sources if "T_H" in hs.__dict__]
        hsT_H.append(10e3)
        self.min_T_H = min(hsT_H)

    @property
    def T_C(self):
        if not self._T_C:
            self._T_C = self.equilibrate()
        return self._T_C

    def __radiator_balance(self, T_C):
        Q_W = sum( [hs.q_w(T_C) for hs in self.heat_sources] )
        Q_S = sum( [surf.direct_solar_thermal_heat_received(self.loc) for surf in self.
            radiators] )
        Q_E = sum( [surf.radiated_heat(T_C) for surf in self.radiators] )
        return Q_E - Q_W - Q_S

    def equilibrate(self, initial_value = 273.15, tol = 1e-3):
        try:
            self._T_C = toms748(self.__radiator_balance, 0, self.min_T_H, rtol = tol)
        except ValueError:
```

```
            raise ValueError("A radiator temperature cannot be found that is less than the minimum T_H in the system. You need a larger, more emissive, or less absorptive radiator. Quitting.")

        for surf in self.radiators:
            surf.set_T(self.T_C)
        for hs in self.heat_sources:
            hs.T_C = self.T_C
        return self.T_C

    def __repr__(self):
        return "heat sources:\n" + "\n\n".join([str(heat_source) for heat_source in self.heat_sources]) + "\n\nradiators:\n" + "\n".join([str(rad) for rad in self.radiators])


if __name__ == "__main__":

    ##--Define the location in space
    loc = Location(distance_from_sun_au = 0.25)

    ##--Thermal Loop 1
    ##----Define power sources:
    ##------Define a nuclear reactor:
    reactor_1   = Reactor(brand = "fission_frontiers", model = "wyvern")

    ##------Define a Solar-Thermal Power system
    absorber_surface_1  = Surface(emission_area = 1000, sun_facing_area = 1000, emissivity = 0.12, absorptance = 0.96, angle_to_sun_deg = 25)
    absorber_surface_2  = Surface(emission_area = 200, sun_facing_area = 200, emissivity = 0.12, absorptance = 0.96, angle_to_sun_deg = 20)
    solar_thermal_1     = SolarAbsorber(loc, 1e6, [absorber_surface_1, absorber_surface_2])

    ##----Define the shared radiators
    radiator_1  = Surface(2 * 1000**2, 1000**2, 0.9, 0.09, 90)
    radiator_2  = Surface(2 * 800**2, 800**2, 0.9, 0.09, 70, temperature_offset = -20)
    radiator_3  = Surface(2 * 800**2, 800**2, 0.9, 0.09, 70, temperature_offset = -20)

    ##----Assemble a thermal-loop
    thermal_loop_1 = ThermalLoop(loc, heat_sources = [reactor_1, solar_thermal_1], radiators = [radiator_1, radiator_2, radiator_3])
    thermal_loop_1.equilibrate()

    ##--Printing out the standard (rather inconvenient) format for a thermal loop object
    print("thermal_loop_1:")
    print(thermal_loop_1)

    ##--Thermal Loop 2
    ##----Define power sources for Thermal Loop 2:
    pv_surface_1    = Surface(2 * 100, 100, 0.92, 0.12)
    photovoltaic_1  = Photovoltaic(loc, 0.33, 25 + 273.25, 0, pv_surface_1)
```

```
##--The photovoltaic panel surface will act as its own radiator to an extent, but it is
    still necessary to tell ThermalLoop that it will act thus by passing it in.
##--You can also add additional radiators to your solar panels, for example via an active
    cooling circuit. We include a negative temperature offset to represent that the panels
     will be hotter than an external radiator since the heat will have to be moved from
    the panel to the radiators
pv_rad = Surface(2000, 1000, 0.78, 0.78, 90, temperature_offset = -10)

thermal_loop_2 = ThermalLoop(loc, heat_sources = photovoltaic_1, radiators = [pv_surface_1
    , pv_rad])
thermal_loop_2.equilibrate()

##--You can also print out specific objects to make the data easier to understand
print("photovoltaic_1:")
print(photovoltaic_1)
print()

print("pv_rad:")
print(pv_rad)
print()

##--Operational Loop 1
atmospheric_heat_load_1 = HeatSource(100000)
atmospheric_heat_load_2 = HeatSource(200000)

operational_loop_rad_1 = Surface(2000, 1000, 0.92, 0.12, angle_to_sun_deg = 90,
    temperature_offset = -20) ##--The sun facing area is half, but will actually fall to
    zero because it is at 90 degrees to the sun

operational_loop_1 = ThermalLoop(loc, [atmospheric_heat_load_1, atmospheric_heat_load_2],
    [operational_loop_rad_1])
operational_loop_1.equilibrate()

print("operational_loop_1:")
print(operational_loop_1)
##--We can see that this radiator is too big for this heat load, as the Cold-Side
    Temperature perceived by the heat sources is 251.6 K (-23.6 degC) - a little chilly
    for your residents!
```