```
clearscreen.
print "green light".


FUNCTION heading_of_vector {
  PARAMETER vecT.
  LOCAL east IS VCRS(SHIP:UP:VECTOR, SHIP:NORTH:VECTOR).
  LOCAL trig_x IS VDOT(SHIP:NORTH:VECTOR, vecT).
  LOCAL trig_y IS VDOT(east, vecT).
  LOCAL result IS ARCTAN2(trig_y, trig_x).
  IF result < 0 {RETURN 360 + result.} ELSE {RETURN result.}
}

// Get the heading of the ship's nose (where it's pointing)




set initial_part_count to SHIP:PARTS:length.
wait UNTIL SHIP:PARTS:length < (initial_part_count - 1).
rcs on.
set SHIP:CONTROL:YAW to -1.
wait 1.

ag1 on.
set throt to 0.5.
lock throttle to throt.




SET shipPointingHeading TO heading_of_vector(SHIP:FACING:FOREVECTOR).
until shipPointingHeading > 180 {
    SET shipPointingHeading TO heading_of_vector(SHIP:FACING:FOREVECTOR).
    print shipPointingHeading.
}

set SHIP:CONTROL:YAW to 0.
```

```
set steeringmanager:rollcontrolanglerange to 0.
set SteeringManager:ROLLTORQUEFACTOR to 100.
PRINT "Second stage ignited!".
// Define a function to predict impact position

set throt to 1.

set finDir2 to 280.
set pit to -5.
lock steering to heading(finDir2,pit).
set throt to 1.
ag1 on.
lock throttle to throt.


set distance to 500.

set rfLat to -0.0882343165971148.
set rfLng to -74.5930087859275.


set impactLng to rfLng + 1.
set timeStep to 0.5.
set kerb to ship:body.
set timeOFFset to 0.

set currentPos to ship:position.
if kerb:ALTITUDEOF(POSITIONAT(SHIP, TIME:SECONDS + timeOffset)) > 0 {
    until kerb:ALTITUDEOF(POSITIONAT(SHIP, TIME:SECONDS + timeOffset)) < 0 {
        SET timeOffset TO timeOffset + timeStep.
```

```
    }
} else {
    until kerb:ALTITUDEOF(POSITIONAT(SHIP, TIME:SECONDS + timeOffset)) > 0 {
        SET timeOffset TO timeOffset - timeStep.
    }
    SET timeOffset TO timeOffset + timeStep.
}
clearscreen.

set finalPos to POSITIONAT(SHIP, TIME:SECONDS + timeOffset).
set impactPos to kerb:GEOPOSITIONOF(finalPos).
set impactLat to impactPos:lat.
set impactLng to impactPos:lng - (360 / kerb:ROTATIONPERIOD) * timeOffset.

set tarLng to (rfLng-impactLng).
set tarLat to (rfLat-impactLat).
set sMag to sqrt(tarLng^2 + tarLat^2).




until impactLng < rfLng {
    set currentPos to ship:position.
    if kerb:ALTITUDEOF(POSITIONAT(SHIP, TIME:SECONDS + timeOffset)) > 0 {
        until kerb:ALTITUDEOF(POSITIONAT(SHIP, TIME:SECONDS + timeOffset)) < 0 {
            SET timeOffset TO timeOffset + timeStep.
        }
    } else {
        until kerb:ALTITUDEOF(POSITIONAT(SHIP, TIME:SECONDS + timeOffset)) > 0 {
            SET timeOffset TO timeOffset - timeStep.
        }
        SET timeOffset TO timeOffset + timeStep.
    }
    clearscreen.

    set finalPos to POSITIONAT(SHIP, TIME:SECONDS + timeOffset).
    set impactPos to kerb:GEOPOSITIONOF(finalPos).
```

```
    set impactLat to impactPos:lat.
    set impactLng to impactPos:lng - (360 / kerb:ROTATIONPERIOD) * timeOffset.
    print timeOffset.



    set tarLng to (rfLng-impactLng).
    set tarLat to (rfLat-impactLat).
    set cMag to sqrt(tarLng^2 + tarLat^2).
    if cMag < sMag * 0.2 {
        set throt to 0.4.
    }



//-------------------------------------------------------------------------------
-----------



    set finLat to rfLat - impactLat.
    set finLng to rfLng - impactLng.


    if finLat > 0 {
        set TarDirAdd to 180. //SOUTH


    }else if finLat < 0 and finLng > 0 {
        set TarDirAdd to 0. // NORTH West


    }else if finLat < 0 and finLng < 0 {
        set TarDirAdd to 360. // NORTH EAST


    }
    set finDir to TarDirAdd + 180 + arctan((finLng)/(finLat)).
    if finDir > 360 {
        set finDir to finDir - 360.
```

```
    } else if finDir < 0 {
        set finDir to finDir + 360.
    }

    print finDir.
    set finDir2 to finDir.
    print " ---------------------- ".

    print ship:groundspeed.


}

lock throttle to 0.

lock steering to ship:SRFRETROGRADE.
wait until ship:verticalspeed < -200.



set sAlt to 85.
//this current value is specific to my vehicle, needs to be changed.


set tarCon to 3800.              // target constant (the prefix tar is used in all
variables pertaining to target vector)
set shipCon to 1.3.             //ship constant


set tarConFall to -2700.
set shipConFall to -1.9.




ag3 on.
```

```
set sLon to -74.5577303699591.
set sLat to -0.0949776193834427..

brakes on.
set steeringmanager:rollcontrolanglerange to 0.
set SteeringManager:ROLLTORQUEFACTOR to 0.
print availablethrust.
set TTW to availablethrust/(mass*9.8).
set LTTWR to 0.
until ship:altitude < 1200 {


    if ship:altitude < 3500 {
        if (ship:mass*(ship:verticalspeed^2/(altitude-sAlt)))/availablethrust > 1
{
            set LTTWR to 1.
        }
    }
    if ship:altitude < 25000 {
        rcs off.
    }


    set pLon to ship:geoposition:LNG.
    set pLat to ship:geoposition:LAT.
    set dTarget to
sqrt((((ship:geoposition:LNG-sLon)*10471)^2+((ship:geoposition:LAT-sLat)*10471)^2
)).


    wait 0.02.
    set tarLng to (sLon-ship:geoposition:LNG)*tarConFall.
    set tarLat to (sLat-ship:geoposition:LAT)*tarConFall.
    set tarMag to sqrt(tarLng^2 + tarLat^2).

    if tarMag > 120 {
        set tarLng to tarLng * 1.
```

```
    set tarLat to tarLat * 1.
}
print tarMag.


set shipLng to (pLon - ship:geoposition:LNG).
set shipLat to (pLat - ship:geoposition:LAT).
set shipMag to sqrt(shipLng^2 + shipLat^2).
set diff to ship:groundspeed / shipMag.
set shipLng to (shipConFall*shipLng*diff).
set shipLat to (shipConFall*shipLat*diff).
set shipMag to sqrt(shipLng^2 + shipLat^2).


//FINAL VECTOR CACULATION

set finLng to tarLng + shipLng.
set finLat to tarLat + shipLat.
set finMag to sqrt(finLng^2 + finLat^2).
set lPitch to 90 - finMag.
if lPitch < 45 {
    set lPitch to 45.
}


if finLat > 0 {
    set TarDirAdd to 180. //SOUTH


}else if finLat < 0 and finLng > 0 {
    set TarDirAdd to 0. // NORTH West


}else if finLat < 0 and finLng < 0 {
    set TarDirAdd to 360. // NORTH EAST


}
set finDir to TarDirAdd + 180 + arctan((finLng)/(finLat)).
```

```
    if finDir > 360 {
        set finDir to finDir - 360.
    } else if finDir < 0 {
        set finDir to finDir + 360.
    }


    lock steering to heading(finDir,lPitch).
}




//SUICIDE BURN LOOP
//this next section is very similar to the last, because it used the same
fundamental code with different constants.

set throt to 0.
lock throttle to throt.
ag2 on.
brakes off.
rcs on.
set pAltitude to ship:altitude.
wait 0.1.
until ship:verticalspeed > 0 {
    set pAltitude to ship:altitude.
    set steeringmanager:rollcontrolanglerange to 0.
    set SteeringManager:ROLLTORQUEFACTOR to 100.

    set pLon to ship:geoposition:LNG. //this is needed for the ship vector
calculation, it explains what pLon
    set pLat to ship:geoposition:LAT. //and pLat are in the ship vector section


    wait 0.02.      //this seems to help the flow


    if ship:altitude > 800 {
        set throt to 0.
```

```
    } else {
        if ship:verticalspeed < -130 {
            set throt to .9.
        } else {
            ag4 on.
            gear on.
            set throt to
(ship:mass*(ship:verticalspeed^2/(altitude-sAlt)))/availablethrust.
        }
    }




    set tarLng to (sLon-ship:geoposition:LNG)*tarCon.
    set tarLat to (sLat-ship:geoposition:LAT)*tarCon.
    set tarMag to sqrt(tarLng^2 + tarLat^2).
    if tarMag > 35 {                                    //This if statement
basically limits it to 20 degrees of
        set tarDiff to 35/tarMag.                       //pitch, so that if it
lands really far from the target
        if ship:altitude < sAlt + 60 {                  //it doesnt just flip the
ship over completely.
            set tarDiff to 0.                           //it also turns this
vector off completely when its less
        }                                               //than 30 meters above
the target. So it gives up on
        set tarLng to tarLng * tarDiff.                 //trying to land on
target at that altitude and focuses
        set tarLat to tarLat * tarDiff.                 //on trying to kill off
all groundspeed
    }
    set tarMag to sqrt(tarLng^2 + tarLat^2).




    set shipLng to (pLon - ship:geoposition:LNG).
```

```
    set shipLat to (pLat - ship:geoposition:LAT).
    set shipMag to sqrt(shipLng^2 + shipLat^2).
    set diff to ship:groundspeed / shipMag.             //This parts hard to
explain, it basically makes the magnitude
    set shipLng to (shipCon*shipLng*diff).              //of the vector tied to
the ship groundspeed, just leave this
    set shipLat to (shipCon*shipLat*diff).              //part in.
    set shipMag to sqrt(shipLng^2 + shipLat^2).




    set finLng to tarLng + shipLng.
    set finLat to tarLat + shipLat.
    set finMag to sqrt(finLng^2 + finLat^2).
    set lPitch to 90 - finMag.
    if lPitch < 70 {
        set lPitch to 70.
    }


    if finLat > 0 {
        set TarDirAdd to 180. //SOUTH


    }else if finLat < 0 and finLng > 0 {
        set TarDirAdd to 0. // NORTH West


    }else if finLat < 0 and finLng < 0 {
        set TarDirAdd to 360. // NORTH EAST


    }
    set finDir to TarDirAdd + 180 + arctan((finLng)/(finLat)).
    if finDir > 360 {
        set finDir to finDir - 360.
    } else if finDir < 0 {
        set finDir to finDir + 360.
```

```
    }
    set steeringmanager:ROLLPID:KP to 0.
    set steeringmanager:ROLLPID:KI to 0.
    SET SHIP:CONTROL:roll to 0.00001.
    lock steering to heading(finDir,lPitch).
    clearscreen.
    print ship:verticalspeed.
}
clearscreen.
lock throttle to 0.
print " testing2 done".
wait 3.
rcs off.
unlock steering.
```