# CS4303 Practical 4

# Terraform

# 160012340

## Introduction

In this practical I implement the game I designed, called Terraform.

## Design and Implementation

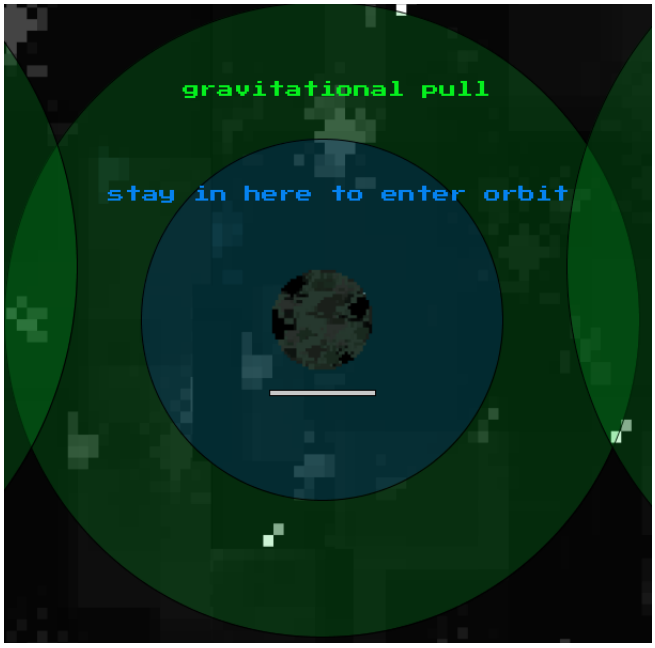
As described in my pitch, the game is a physics based 'drag and launch' adventure set in space. Originally it was designed as a vertical shmup type of game, where the planets scroll down the screen and the users goal is to collect points without falling off the screen. Also in the initial design was a battle stage. In my feedback it was pointed out that it may be fun to have the battle and alien interactions happen while the user travelled the map.
I toyed around with this idea during development and ended up deciding to make the game take place on a single non moving screen, and have aliens to deal with while traversing the planets.
As a result of this I knew I would also have to design an interesting procedural generation algorithm to make each level, challenging, fun, and progressively harder than the last while still completable.

### Movement and Physics

I designed the movement to be drag and launch, so to move the player must click and drag to decide the direction and power the ship will be launched at. I made this in reference to the ships position, but the user does not have to begin by clicking on the ship itself. I made the game pause when the launching is being decided, this makes the game a lot more enjoyable as it can give the user a moment to think when there is a lot happening, and give the user the chance to move the arrow to where they want to click - this is especially useful when not using a mouse.
A majority of the complexity in movement comes from the gravitational forces coming from the planets. While I know everything should have some gravitational pull I limited this effect to the planets as a simplification. I also found the game is more enjoyable when gravity only has an effect from a certain radius away from the planet, this range can be viewed by the user during the game and is another simplification of real life physics. As the user drags to launch, a simulation is done each frame of that launch, which keeps track of the positions of an invisible ship as it is launched with the current drag vector. This allows the user to see where that launch will launch the ship. I experimented for a long time with this to ensure it does not lag the game, and is as computationally inexpensive as possible.

### Terraforming

The aim of the game is to terraform all the planets on each level - that is to turn them from dead rocks to planets full of life. To begin terraforming, the player must remain in the "orbit radius" of a planet for 200 frames. This essentially requires the player to orbit a planet themselves using the physics dynamics and the drag and launch mechanics. A stable orbit is not always possible however, and in these cases the player is simply required to remain in that radius, so for example could move past the planet slowly.




### Fuel Mechanics

The ship has a certain amount of fuel, and when dragging to launch the player can see how much that launch will cost them, depending on the magnitude of the launch. If the player runs out of fuel, they will not be able to launch and will eventually drift into a planet or out into space. Fuel is regained by fully terraforming planets. Fuel may also be lost by being hit by an asteroid.
This mechanic makes the player think a little more strategically about their launching, and conserve fuel.

### Enemies and Obstacles

The aliens in this game are designed to slow progress on the player by taking away the life force of the planets. They are artificial intelligence and can calculate an angle and force to launch themselves at in a similar style to the ship to go to their target planet while being affected by gravity. They choose planets based on both their current life force and their distance from the alien.
Each round there is a meteor shower, these can detract the life force of planets, and damage the players ship - if the player has no fuel or not enough to lose the player will lose the game if hit by a meteor.
If a planet is fully terraformed, it has to lose ALL of it's health to revert to the dead state,

however if not fully terraformed, the planet will revert through it's states as it loses health. This is to make living planets worth defending for the player.

### Laser

The player can shoot a laser (dynamics inspired by the spartan laser from the *Halo* series) to defend themselves and their planets from harm. Once charged this laser can destroy meteors and kill aliens. I didn't want to introduce too many controls for the player to have to remember, so I also made this a left click - while holding 'shift' to differentiate from a launch.

## Procedural Generation

I wanted each level to be enjoyable and interesting to play. I tested out a few procedural generation algorithm including *Poisson-Disc Sampling*[1] to generate random planet layouts for each level. I decided that while this was randomly generated and did not allow for impossible levels it was not interesting enough to play.
I then mapped out some levels that would be interesting and that I wanted to be able to spawn when a level was generated and I realised that they were all vertically symmetrical. This lead me to the algorithm I currently have. For each level, I check if the number of planets is odd. If it is odd and the number is less than 5 I place one planet in the exact centre to ensure symmetry. If the number if 5 or greater, there is a 40% chance that 3 will be spawned along the centre - one in the exact centre, and the other two randomly spawned above and below it. If this does not happen only one is spawned in the centre.

After the centre planet(s) are spawned, half of the remaining planets are randomly spawned to the left of the centre - ensuring they are reasonably spaced and not too close to the players spawn point. I have defined too close as a planet being totally encased by the other planets radius for gravitational pull - as this would logically mean in the game that the planet may be affected by that gravity. Outside of logic, this ensure that there is some way of the player orbiting that planet long enough to enter the terraforming animation.

After placing these planets my *symmetry()* function spawns the rest on the right side of the centre. This leaves each map totally symmetrical along the vertical axis, which in my opinion is a nice degree of randomness while also guaranteeing interesting level design.

[1] https://www.jasondavies.com/poisson-disc/

## AI Alien

### Planet targeting

The alien is designed to attack planets. When deciding which planet to attack it assigns all planets a target value using the following equation:

$$targetVal = 2 * lifeForce + D$$

Where D is the distance from the alien to the planet. It then chooses the planet with the highest target value using a find max loop. I chose to incorporate distance as the alien can only do damage when actually at the planet, so the time it takes to get there is definitely a factor in how troublesome it can be for the player. The Aliens will work on a team, and will not choose a planet which is currently being target by another Alien, this is another feature to make them more troublesome for the player.

### Path finding

When moving from a current position, the aliens only need a single launch to reach the planet. The function to calculated the exact initial velocity (in both direction and magnitude) needed to launch itself there is as follows:

**Algorithm 1:** Alien pathfinding

```
Result: Magnitude and Direction of Initial Velocity
start = vector from subtraction of current position from target planet;
sightLength = distance from current position to target planets;
clockwise = random true or false;
found = false;
magCount = 0;
while !found do
    if magCount ¡ 12 then
        increase magnitude of vector;
        magCount++;
    else
        magCount = 0;
        reset magnitude of vector;
        if clockwise then
            turn vector by π/8;
        else
            turn vector by -π/8;
        end
    end
    found = simulated path reaches target planet within sightLength integrations;
end
launch
```

Similar to the ships drag and launch projection, each step the alien simulates a flight path by integrating and updating the gravitation pulls sever times, and tracking the position vector as it goes. As can be seen, steps are taken to ensure this is computationally efficient. The simulation is only done to the length of the distance between the alien and the target, the launch vector is turned by $\frac{\pi}{8}$ so not every angle is checked. I also wanted to ensure that this simulation didn't lag the game, so I designed it such that one one simulation is done per frame, and the alien waits and "thinks" until a path is found - although this is generally still very quick. If a path is not found the alien chooses a new target planet.

When moving from one planet to the next, the alien begins by elimination the large gravitational force of the plan it is on by launching itself a distance away from that planet before calculating it's next route. This was somewhat for my own convenience and also to make the game a little easier for the player, as this thinking period away from the target is an ideal time to shoot for a strategic user.

The path finding for this took a long time to get right, and make computationally efficient - I did consider and experiment with other algorithms such as A*, though these were not as suitable given the effect of the other planets, and that this is not calculating a route where each step is decided, but the features of one blast which decides a whole route.

### Decision Making

The alien is generally either searching for a new target, travelling to that target, or consuming the life force of a target. It decides when to leave that planet under certain conditions:

- It has to have been on that planet for at least x number of frames (randomly decided between 100 and 700)
  - this is to ensure the alien spends some time on each planet, and does not planet hop too quickly, as this makes game play less enjoyable.
- any of the following:
  - The planets life force has reached 0.
  - The player is immediately nearby.
  - The player is somewhat nearby, and is moving in roughly the direction of the alien.

The alien is evasive of the player, and through the dot product of the vector between the aliens position and the players position, and the ships velocity, can tell which direction the player is headed. This allows it time to move away from a planet if the player is on their way, and evade the player.

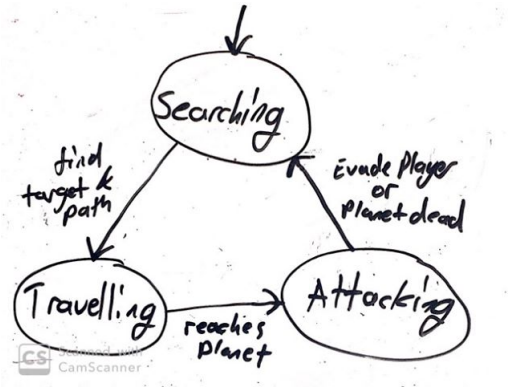



Figure 1: Decision tree for the alien AI

## Spawning

The player always spawns the in the same place, and is not affected by the gravitational pull of any planets - this is to let the user get their bearings at the beginning of each level.

During meteor showers the asteroids are procedurally generated, with a 1% chance of spawning a single asteroid each frame. They randomly choose a side of the screen, and randomly choose a position along that side, and are spawned there.

The alien is spawned similarly to the asteroids, randomly choosing a side of the screen, and then randomly choosing a position along that side to spawn.

The game is made increasingly difficult as it goes on. As the player progresses through the levels, more aliens and asteroids are spawned, there are more planets to traverse, which in turn makes any movement harder as the gravitational effects become more complex.

## Physics

The most important aspect of this game is its physics, the action of gravity is the players' only tool for navigation and completely determines the trajectory of the ship around the planets.

### Drag and Launch

The initial thrust of the ship is determined by the player. When the player pulls back their mouse a line is drawn where the length determines the thrust of the ship, the thrust applied is a value proportional to that length. The direction of thrust is applied in a direction opposite to that drawn by the mouse. When the force is applied, it is applied instantaneously and will give the ship some initial velocity that it is propelled with. This initial velocity is determined by two factors: the thrust applied by the player and the mass of the fuel in the ship.

Throughout each level the ship loses fuel with each launch and the law of conservation of momentum says that if the mass of the ship decreases (as it will with a reduction of fuel) then the velocity must increase. To incorporate this effect into the game play, the **Tsiolkovsky ship equation** was used.[1] This equation determines how the velocity of a ship will change as its mass changes, it is:

$$\Delta v = v_e \ln\left(\frac{m_R + m_{fT}}{m_R + m_{fC}}\right),$$

where $\Delta v$ gives the amount the velocity will change by, $v_e$ is the velocity due to the thrust, $m_R$ is the mass of the ship, and $m_{fT}$ and $m_{fC}$ are the masses of fuel full and current fuel respectively. The initial velocity given to the ship after each click is then given by $v = v_e + \Delta v$. After launch, the ship is affected by Newtons' first law which states that the ship will continue in the same direction with the same velocity given by the propulsion unless no other forces are acting upon it (such as gravitational pull).

When the player launches, the previous velocity of the ship is disregarded, and the ship is treated as if it is moving from a complete stand still. While not accurate reality this tweak to the physics I felt provided a good level of challenge to the game, and ultimately made it more enjoyable.

### Orbital Mechanics

Once the ship has been launched by the user, the gravitational forces between the ship and the planets determine the motion. These forces are governed by **Newtons' law of universal gravitation**, which is:

$$F = G\frac{m_1 m_2}{r^2},$$

where $G = 6.67 \times 10^{-11}$ N $\cdot$ (m/kg)$^2$ is the "gravitational constant", $m_1$ and $m_2$ are the masses of the objects, and $r$ is the distance between them.[2] The equation in the current form only allows forces to be calculated between two objects and doesn't give any directions, so we can generalise it to find the force on the ship by an arbitrary number of planets $N$ and in vector form to give directions. It is stated as,

$$\mathbf{F} = -Gm_R \sum_{i=1}^{N} m_i \frac{\mathbf{r}_0 - \mathbf{r}_i}{|\mathbf{r}_0 - \mathbf{r}_i|^2},$$

where the bold vectors represent the PVectors I used, $r_0$ is the position of the ship, and $r_i$ is the position of the $i$th planet. This equation gives me the PVector to be used in integration, to integrate to get each position of the ship per frame.

### Planets and Sphere of Influence

The planets themselves have had masses set by realistic values from space itself, after some experimentation I found that the mass of one of Jupiter's moons gave rise to the best orbital mechanics for game play when the ship's mass was set to simply 1. I chose to make the mass of the planet that of Jupiter's moon Europa.[3]

## Scoring, losing and winning.

The player gains points as they progress through the planets. Points are scored by fully terraforming planets, shooting aliens and asteroids, and completing levels.
The player can lose by dying in one of three ways:

- Crashing into a planet
- Lost in space (off screen)
- Hit by asteroid/meteor with low/no fuel.

The player wins the game by surviving 10 levels.
The aim of the game is to get to the end with the highest score possible.

### Sound

I used the sound library to include sound effects which make events in the game more noticeable and understandable. The sounds were all from *freesound.org* and the links to each sound are in the code.

### Graphics

As can be seen, the game has an 8-bit style. As it was relatively simple I drew the graphics themselves and use the *.png*'s as sprites. While I know this isn't a large factor for marking I felt it made the game more enjoyable to play, and made the mechanics a lot easier to understand for the player. I tried to keep the graphics similar to those I made for the initial pitch.

## Video

The video can be viewed here:

https://vimeo.com/381361867
The password to view is "cs4303"
It got cut off for some reason, given more time I would have liked to fix this - but the gameplay is still fully viewable.

## Context

## Context

The initial idea for the drag and launch vertical jump was inspired by a mobile game called "Dash Valley" which as the user drag and launch a circle through an obstacle course, where the world slows down as the user drags.

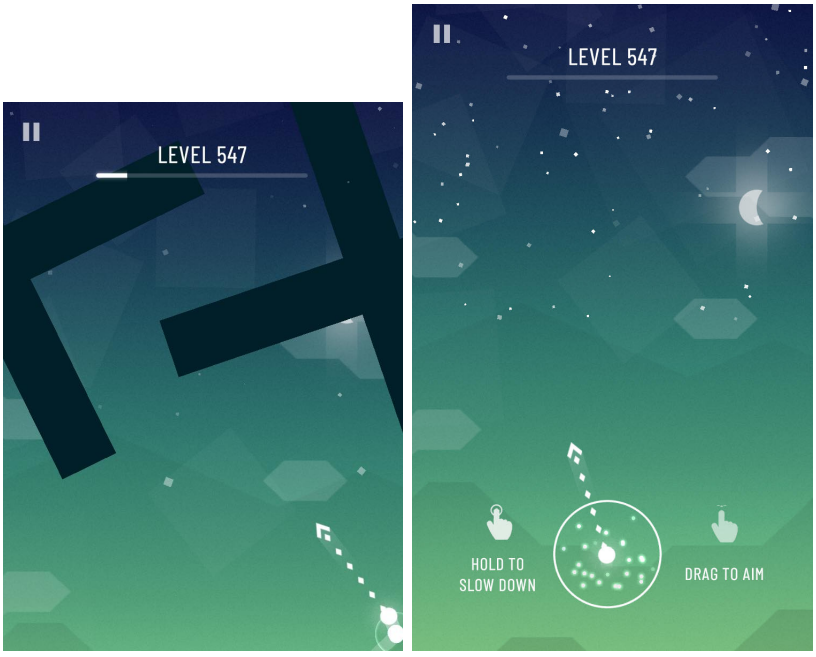



Figure 2: Dash Valley Drag and Launch Mecahnics

The orbital mechanics weren't really inspired by any game, but are comparable to the space levels of Angry Birds Space, which is also drag and launch:


Figure 3: Angry Birds mechanics

My game is indeed comparable to these games in terms of the drag and launch dynamic (which I certainly did not invent), and the gravitational pull within a radius in the case of Angry Birds, but does have it's own twists. It incorporates these elements with the sense of an RTS style game when there are Aliens working against you, and fuel is a restriction, requiring strategy to get through the levels. The procedural generation of the levels and the different types of planets possible give it a sense of adventure as well.

## Evaluation, User Feedback and Conclusion

I used an online form to have people beta test and rate the game in several areas, the feedback I got was as follows:

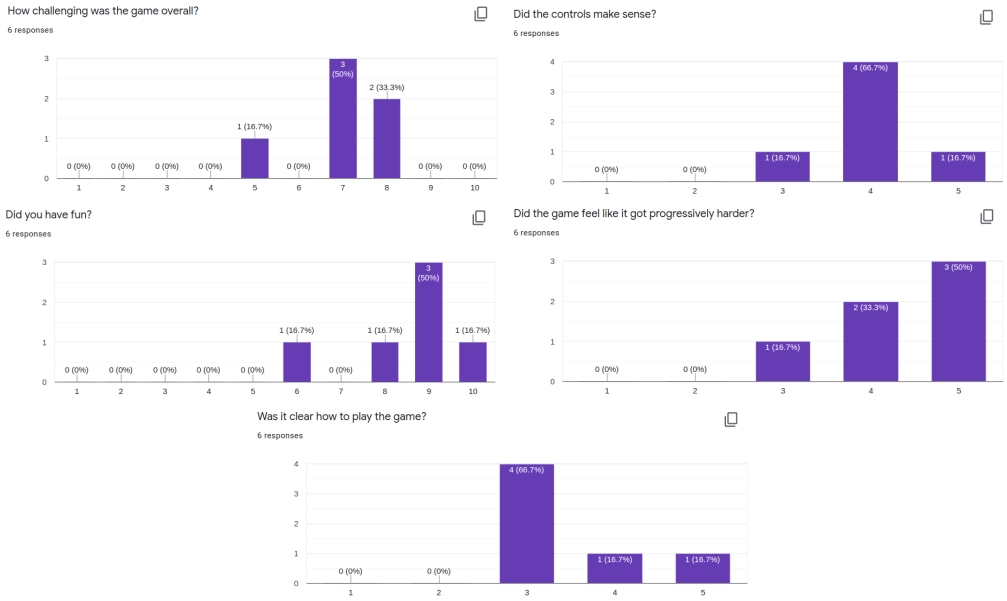



Figure 4: User Feedback

Following this feedback I made the game somewhat easier to play by lengthening the projection line, and slowing down the asteroids. I also added a option that shows the user how to play during the game should they want to check anything.

Other changes I made due to feedback are as follows:

- Asteroids were initially an instant death, I changed this to damage fuel initially.
- I show the user why they died on the death screen.
- I made the hit boxes for the aliens slightly larger.

**Conclusion**

I really enjoyed this practical, and I am very happy with the game I have implemented!. I believe I have fulfilled the specification as much as possible, and completed a game which was technically challenging, original, and fun to play. Much like with the last 2 practicals I tried to balance out the randomness, complexity, and how realistic the physics was with how enjoyable it is for the player.

## References


[1] MIT lectures on Variable Mass Systems. J. Peraire, S. Widnall. Accessed on 24/12/19. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec14.pdf

[2] Lecture Notes on Basic Celestial Mechanics. Sergei A. Klioner. 04/09/16. Accessed on 24/12/19. https://arxiv.org/pdf/1609.00915.pdf

[3] Website: The Planets, Moons of Jupiter. Accessed on 24/12/19. https://theplanets.org/moons-of-jupiter/.

[4] Fundamentals of Astrodynamics. Bate, Roger R.; Donald D. Mueller; Jerry E. White (1971). New York: Dover Publications. pp. 333–334.