1 Abstract

The specification outlined the creation of a variant of the classic video game Missile Command through use of the Processing language. The final implementation includes the variations specified, including the use of simple gravity and drag physics simulation. In addition to this, the variant extends upon the original game through the inclusion of an upgrade shop, starting options menu and an extensive class inheritance system to support easier future development of games in Processing.

1.1 Player's Guide

1.1.1 Premise

Protect the planet from assured destruction for as long as you can in Asteroid Defence! Man your cannons and shoot bombs to intercept incoming asteroids! Upgrade your arsenal in the shop to create bigger explosions as you face an evergrowing swarm of asteroids, but beware of more challenging opponents. Shoot down cluster asteroids before they split into multiple lethal fragments and eliminate the satellite before it drops even more asteroids on your cities. Finally, look out for the smart asteroids that can avoid explosions and the bomber that shoots targeted strikes at your cannons and cities.

1.1.2 Controls

Mouse: Aim ballista

z: Switch ballista (descending order)

x: Switch ballista (ascending order)

Space: Explode bomb (Aim: Specific bomb. Mouse at edge of screen: In order fired. Can be changed in options to exclusively use the order in which bombs were fired.)

c: Explode all bombs

Enter: Start game / next wave

2 Game Design

2.1 Audio and Graphics

While initially a design closer to that suggested in the specification was intended, this was later changed. It was difficult to find assets that fit the theme of ballistae, so the design was changed to an aesthetically pleasing, more modern science fiction design. The audio was chosen to match, with inspiration taken from the original Missile Command. The welcome screen can be seen in Figure 1. All audio and graphics were downloaded from https://opengameart.org/, with partial credits available in the game files and the references [1]–[6].

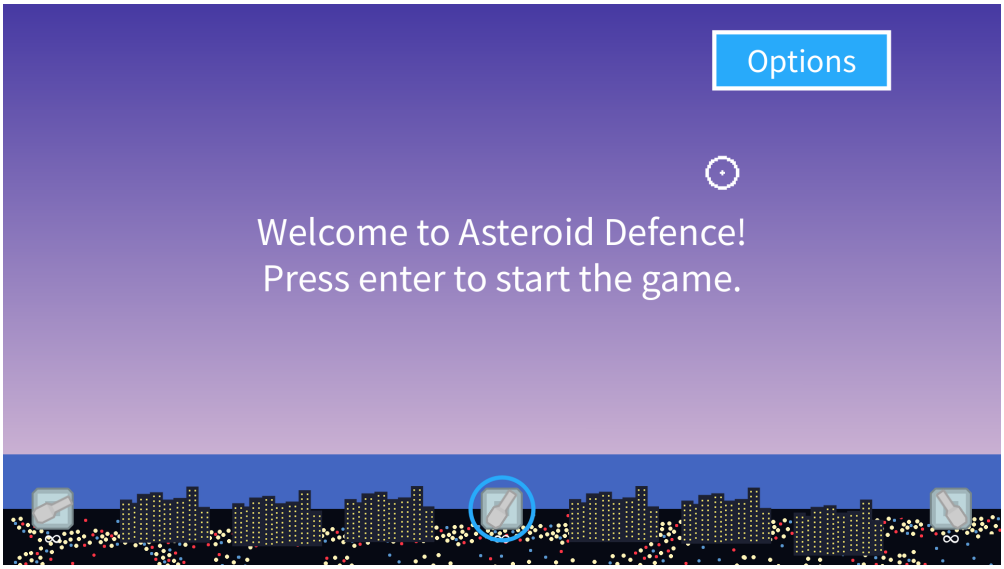


Figure 1: Welcome on Main Menu

2.2 Player

The player is tasked with using bomb-launching ballistae (also referred to as cannons) to protect their cities, starting with 3 and 6 of each respectively on default settings. These numbers can be tweaked in the game settings before starting the game to give the player control of the difficulty.

2.3 Enemy AI

The enemy is computer controlled and aims to destroy the player's ballistae and cities through the use of asteroids. Three asteroid variants and two flying enemy variants are available in game as detailed below and shown in Figure 2.

- Standard Asteroid: Starts at the top of the screen with a random velocity and explodes on contact with the ground or a bomb.
- Cluster Asteroid: Like standard, but may split into two or more fragments along its flight path, giving the potential to hit additional targets.
- Smart Asteroid: Like standard, but can adjust its flight path using a spring force with explosions to avoid them.
- Bomber: Flies across the screen and shoots targeted bombs.
- Satellite: Flies across the screen and drops asteroids randomly.

Figure 2: A swarm of enemies in a game with increased counts

2.4 Waves

The difficulty of each wave is determined by the wave number, which is used in different level parameter calculations.

The number of asteroids is constantly increased across waves, with the time between spawns lowered as waves increase. Furthermore, each asteroid has a chance to spawn as a cluster asteroid with a random split time and size. As waves increase, the chance of being a cluster asteroid is increased in addition to a larger size and lower split time for each one.

Special enemies, including the smart asteroid, bomber and satellite, have their own count per wave. This is randomly chosen, with higher numbers being likelier in later waves. Instead of using a timer, these enemies are spawned more consistently throughout the wave, using the number of asteroids already spawned in as a reference.

2.5 Upgrade Shop

The upgrade shop is available at the end of each wave and allows the player to strengthen their arsenal. Upgrades can be bought through credits mapped from the current score that the player has. They can be bought and sold freely as points permit, with the pause state between waves allowing the player to test any upgrades they buy. Each of the available upgrades is detailed below and shown as in-game in Figure 3.

- Bomb Count: Increases the number of bombs available per ballista to help destroy more asteroids.
- Explosion Size: Increases the radius of bomb explosions to make hitting asteroids easier.
- Nuclear Bomb: The first bombs launched from each ballista as bought result in a much greater explosion.

Figure 3: The shop available between waves

2.6 Options Menu

Before the game starts, the player can choose to access the options menu. This contains settings that allow the player to customise the difficulty of the game. The available settings are listed below and shown as in-game in Figure 4.

- Hybrid Bomb Explosion Order: Whether to use the mouse when selecting a bomb to explode or exclusively in-order explosions.
- Cannon Count: The number of cannons available to the player.
- Starting Ammo: The number of bombs available to the player, excluding shop upgrades.
- Infinite Ammo: Whether to give the player infinite bombs or not.
- City Count: The number of cities to spawn in.
- Credits Multiplier: A multiplier on the number of credits awarded to the player.
- Gravity Multiplier: How strong gravity is on particles.
- Drag Multiplier: How strong drag is on particles.
- Asteroid Spawn Count Multiplier: A multiplier on how many asteroids are spawned each wave.
- Enemy Spawn Rate Multiplier: A multiplier on how quickly asteroids and flying enemies spawn.
- Asteroid Variant Chance Multiplier: A multiplier on how likely asteroids are spawned as variants.
- Flying Enemy Spawn Count Multiplier: A multiplier on how many flying enemies are spawned each wave.

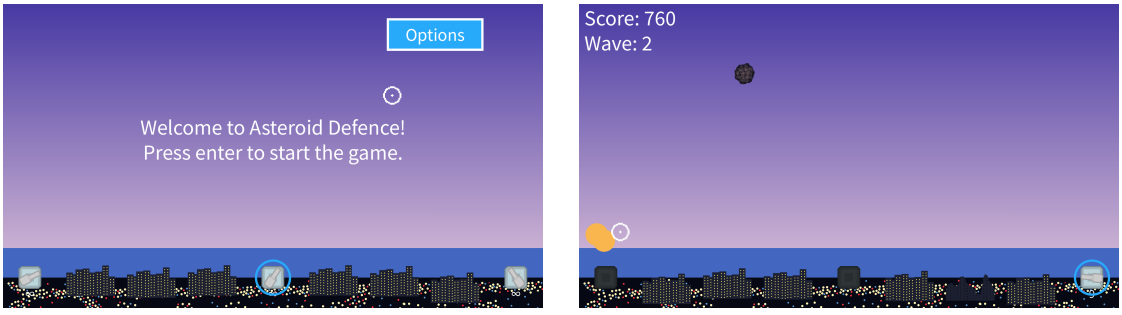
Figure 4: The options menu available on the main menu

3 Implementation

3.1 Level Manager

The 'LevelManager' class was created to manage each part of the game loop. The current state is tracked in a variable, which determines what kind of updates are performed. Initial setup of the play area is performed once during the welcome state. The player is allowed to test the ballistae with infinite ammunition during the welcome and post-level states to get familiar with mechanics and try out upgrades. A summary of each state is listed below with visual examples in Figure 5.

- Welcome: The player is shown a welcome message and can customise options.
- Pre-level: A single-frame state in which the parameters for the next wave are set.
- Level: The main state of the game loop where asteroids are spawned and the player must fight back with limited resources.
- Post-level: The results of a successful wave are shown with the upgrade shop available to the player.
- Game Over: Shows a game over message after all cities have been destroyed and prompts the player to restart.



(a) Welcome State (Main Menu)

(b) Level State

(c) Post-Level State

(d) Game Over State (Main Menu)

Figure 5: The different states of the game loop

3.2 Object Management

To simplify management of object updates and rendering, an abstract 'GameObject' class was created. Each GameObject is given a position in addition to abstract update and render functions. This allows all subclasses to be updated and rendered through a global gameObjects list, rather than requiring further lists and loops for each subclass. A final addition to the GameObject class is a destroyed boolean paired with a getter and setter. This can be called to remove a gameObject from the global list in the next update, allowing it to be garbage collected. Class inheritance is also used more widely across different game elements, such as defining both ballistae and cities as targets and having a base asteroid class that special asteroids extend upon.

The collection type chosen to store most object is the LinkedTransferQueue. This is backed by linked nodes, making insertions constant time. It is also thread-safe, which avoids exceptions from concurrent modification.

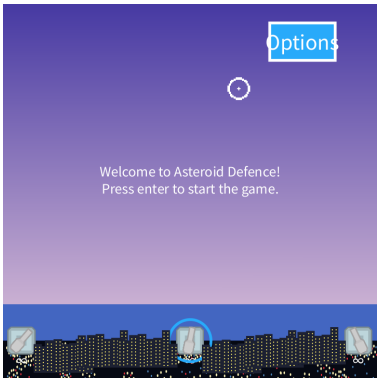
3.3 Player Input

The player is able to control the ballistae using their mouse. These can fire bombs towards the crosshair as long as ammunition remains, which is tracked by using a counter. The ballista to use can be switched at will, but a valid ballista must not be disabled and have ammo remaining. If a ballista is disabled or runs out of ammunition, a switch is performed automatically. Bombs will explode automatically upon contact with an asteroid or another bomb, but can be exploded manually with the press of a key, triggering the bomb closest to the mouse to explode. Advanced players may discover that placing the mouse at the edge of the screen will change the explosion order, with bombs exploded in the order they were launched. These different control options give the player more control over explosion order, allowing for more intricate explosion patterns.

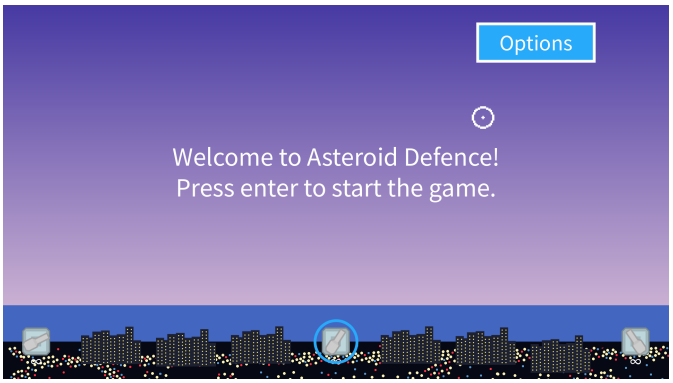
3.4 User Interface and Scaling

Similar to how asteroids, flying enemies and targets have their own base class, the 'UIItem' class was created to serve as a base for buttons and menus. Buttons have their own specialisation in the form of entry, exit, boolean, integer, float and shop buttons. These allow for navigation between menus, adjusting settings and purchasing shop upgrades. The shop and options menus have their own specialisations, but pass the items they contain to the base menu class, which then automatically places them in a rect. This allows for adding and removing of options from menus without having to manually place them. Each button stores its current value that can be accessed publicly to put it into effect.

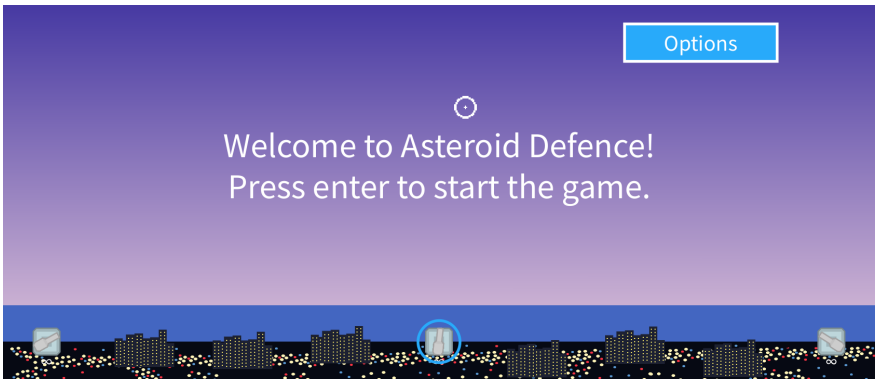
All parts of the game are also scaled by screen size, but using resolutions of a 16:9 aspect ratio are recommended as different ratios may result in different horizontal and vertical scaling as seen in Figure 6.



(a) 1:1 Aspect Ratio



(b) 16:9 Aspect Ratio



(c) 21:9 Aspect Ratio

Figure 6: The game at different aspect ratios

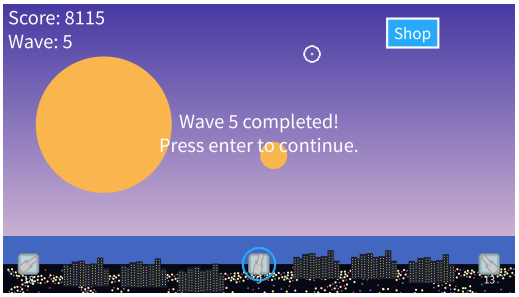
3.5 Explosions and Collision Detection

All explosives explode when falling below the ground height and are deleted when going off-screen on the x-axis. Asteroids do not collide with each other to avoid mid-air explosions when spawning close together. This allows explosive collision code to be deferred to bombs, which iterate through a list of all explosives each frame and check whether the distance between two explosives is less than their radii combined. All explosions similarly check for collisions each frame. Additionally, asteroid explosions not caused by the player check whether any targets are in range at the start of the explosion. A variable held in the 'Explosive' class and passed to the 'Explosion' class is the 'friendly' boolean, which denotes

whether the source of the explosion should do damage to targets or not.

The player is given standard bombs, with larger, nuclear bombs being purchasable in the shop. These are like standard bombs but have their size multiplied and use a different set of graphic and sound. A comparison is shown in Figure 7.

(a): Nuclear Bomb vs Normal Bomb (Unexploded)



(b): Nuclear Bomb vs Normal Bomb (Exploded)

Figure 7: Bomb Variant Comparison

3.6 Points System

As laid out in the specification, the player receives points for surviving cities, remaining ammunition and asteroids exploded. These give a base number of points each that is passed to the level manager's 'addPoints' function, which applies the wave multiplier before adding the points to the score. To ensure that the player is credited only with asteroid explosions triggered via a bomb, the 'friendly' boolean variable discussed in the previous subsection is checked. The score and wave are displayed in the top left corner of the window and appear as seen in Figure 8.

Figure 8: The score and wave display

3.7 Physics Simulation

3.7.1 Managing Forces

The game uses code provided in lectures to simulate each bomb and asteroid as a particle affected by gravity and drag. This involves calculating the force applied to each particle in each update step and then applying this as an acceleration. To ensure updates are consistent across frame rates and screen sizes, updates are decoupled from rendering and any velocities are multiplied by the screen width and height before being applied.

3.7.2 Trajectory Calculation

There are two cases in which the trajectory of a particle must be calculated: When a smart asteroid or bomber missile is spawned falling down onto a target and when the

player shoots a bomb to their crosshair. These are highlighted in Figure 9.

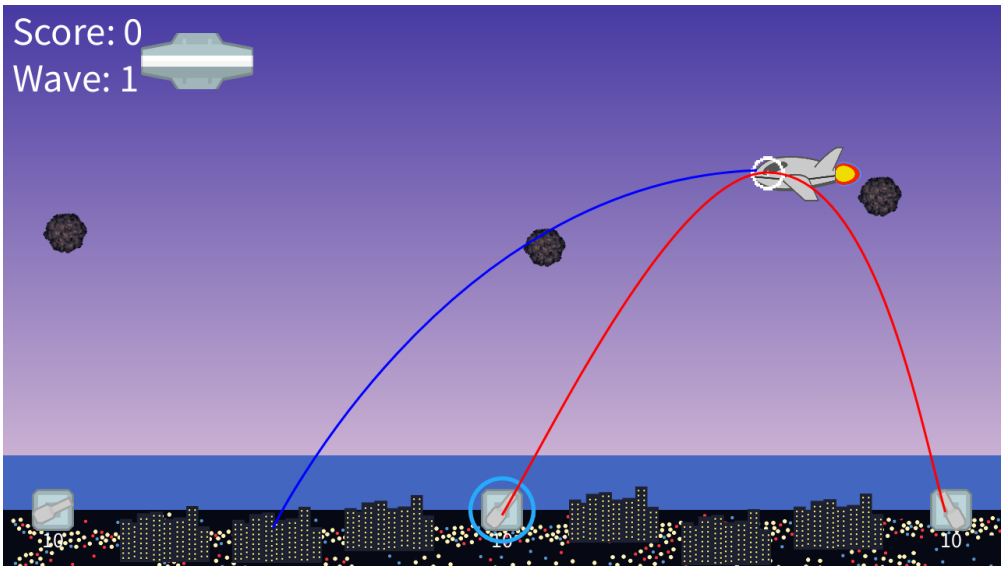


Figure 9: The rising arc (red) and falling arc (blue)

To ensure that each bomb's trajectory lines up with the player's crosshair, the necessary starting velocity must be found. This requires taking into account all forces that act on the bomb during flight. While an attempt was made to fully take drag into account during calculations, this was unsuccessful. This attempt can be found in the appendix. Instead, the trajectory for the case in which there is only gravity is first calculated using the equations of motion as shown below. Then, if drag is active, a constant is applied to counter it. This method unfortunately only works well for the standard drag value.

$$
\begin{aligned}
& v_y(t) = 0 \\
\implies & v_y(0) = -\sqrt{2 \times g \times r_y} \\
\implies & t = \frac{v_y(0)}{g} \\
\implies & v_x(0) = \frac{r_x}{t}
\end{aligned}
$$

4 Testing

The game was playtested to help build a better balance and discover bugs that a player may encounter. This involved playing several rounds with different options and shop upgrades to ensure each feature works as expected. During development, spawn chances of cluster asteroids and special enemies were increased to study their behaviour in closer detail. The addition of options to customise these parameters gives players more control in case they do not like the balance.

One significant unresolved issue lies with the Sound package of Processing, which was used to add sound effects to the game. Playing a large number of sound effects in short succession can result in audio delay and corruption. This may be a result of Processing's audio output latency.

5 Evaluation

The variety of options and shop items available in the game in addition to the base experience give the game significant replayability and customisation to different types of players.

However, several areas that future development could improve were identified. These are listed below.

- Asteroid Targeting Heuristic: Asteroids spawn at the top of the screen with a random x position and velocity. This can result in asteroids flying off-screen and not being a threat to the player if a lot of their cities have already been destroyed. A heuristic to encourage spawns closer to targets could help.
- Sounds: In addition to the previously discussed sound corruptions, the game could benefit from additional sounds such as an indicator for low ammunition as well as sounds played when satellites and bombers are on screen.
- Consistent and clear graphics: The use of graphics created by a large number of people means that these do not all fit well together. Furthermore, near-identical appearance of bomber and player bombs could confuse the player.
- Higher quality graphics: The graphics could be improved, such as through an animated flame behind the bomber and use of an image for explosions. The scaling of cluster asteroid graphic fragments is also poor, with pixelation visible.
- Scaling to different aspect ratios: Scaling is not perfectly consistent across aspect ratios and there is no easy way to force use of the recommended 16:9 aspect ratio.
- Menu scaling: While items in menus are scaled and placed to fit well, the game struggles fitting a large number of items such as seen in the options menu.
- Physics Simulation: While bombs are aligned properly with the crosshair when there is no drag, the alignment is not perfect when drag is switched on. Furthermore, the use of the trajectory peak for calculation means bombs fired near the horizontal behave much differently to those at other angles. This can be an interesting element once the player has gotten used to it, but may be unappealing over a more consistent system.
- Code Quality: While sensible variable names were used to make code more readable, there is a lack of comments detailing the code.

6 Conclusion

The final implementation provides a well-developed variant of the game Missile Command, together with additional features not found in the original to allow for more varied gameplay. While more polish is possible in areas of game design and bug fixing, the extensive class inheritance system implemented serves as a strong base for further development of Asteroid Defence and other games in Processing.

References

[1] software_atelier. 'City pixel tileset.' (), [Online]. Available: https://opengameart.org/content/city-pixel-tileset (visited on 08/02/2024).

[2] qubodup. 'Space jet side sprite.' (), [Online]. Available: https://opengameart.org/content/space-jet-side-sprite (visited on 09/02/2024).

[3] surt. 'Shmup ships.' (), [Online]. Available: https://opengameart.org/content/shmup-ships (visited on 09/02/2024).

[4] MSavioti. 'Missile 32x32.' (), [Online]. Available: https://opengameart.org/content/missile-32x32 (visited on 09/02/2024).

[5] Kenney. 'Space shooter extension 2.50.' (), [Online]. Available: https://opengameart.org/content/space-shooter-extension-250 (visited on 09/02/2024).

[6] Calinou. 'Simple crosshairs.' (), [Online]. Available: https://opengameart.org/content/simple-crosshairs (visited on 10/02/2024).

7 Appendix

7.1 Partial Initial Velocity Calculation With Drag

To ensure that each bomb's trajectory lines up with the player's crosshair, the necessary starting velocity must be found. This requires taking into account all forces that act on the bomb during flight. To find the required starting velocity, we can integrate the projectile's acceleration, which we can find from the forces acting on it.

$$\vec{F}_{net} = m\vec{a}_{net} \Rightarrow \vec{v_{net}} = \int \frac{\vec{F}_{net}}{m} dt$$

In the game, two forces are simulated. The first is a force on the projectile from drag and the second a force from gravity.

$$\vec{F}_{net} = \vec{F}_{drag} + \vec{F}_{gravity}$$

The drag is modelled linearly as follows to simplify integration.

$$\vec{F}_{drag} = -k_1|v|\vec{\hat{v}}$$

The velocity's unit vector can be split up into components as follows.

$$\vec{\hat{v}} = \frac{\vec{v}}{|v|} = \frac{v_x\vec{\hat{x}} + v_y\vec{\hat{y}}}{|v|}$$

Substituting this back into the drag force equation, we get the following.

$$\vec{F}_{drag} = -k_1|v|(\frac{v_x\vec{\hat{x}} + v_y\vec{\hat{y}}}{|v|}) = -k_1v_x\vec{\hat{x}} - k_1v_y\vec{\hat{y}}$$

The force due to gravity is modelled as follows.

$$\vec{F}_{gravity} = -mg\vec{\hat{y}}$$

Putting these two together, we get the net force split into components.

$$\vec{F}_{net} = -k_1v_x\vec{\hat{x}} - (k_1v_y + mg)\,\vec{\hat{y}}$$

We can divide by the mass to get an expression for the net acceleration.

$$\vec{a}_{net} = \frac{\vec{F}_{net}}{m} = \frac{-k_1v_x\vec{\hat{x}} - (k_1v_y + mg)\,\vec{\hat{y}}}{m} = -\frac{k_1}{m}v_x\vec{\hat{x}} - \left(\frac{k_1}{m}v_y + g\right)\vec{\hat{y}}$$

Setting a constant for k_1/m simplifies the expression further.

$$\alpha = \frac{k_1}{m} \Rightarrow \vec{a}_{net} = -\alpha v_x\vec{\hat{x}} - (\alpha v_y + g)\,\vec{\hat{y}}$$

Now we can perform the integration.

$$\begin{aligned}
\vec{v_{net}} &= \int \frac{\vec{F}_{net}}{m} dt \\
&= \int \left(-\alpha v_x\vec{\hat{x}} - (\alpha v_y + g)\,\vec{\hat{y}}\right) dt \\
&= -\alpha\vec{\hat{x}} \int v_x dt - \vec{\hat{y}} \int (\alpha v_y + g)\, dt \\
&= (-\alpha r_x + c_x)\,\vec{\hat{x}} + (-\alpha r_y - gt + c_y)\,\vec{\hat{y}} \\
&= v_x\vec{\hat{x}} + v_y\vec{\hat{y}}
\end{aligned}$$

If we take the launch position to be at the origin, meaning $t = 0$ and $r_x(0) = r_y(0) = 0$, we find that the starting velocity components $v_x(0)$ and $v_y(0)$ are equal to the constants of integration c_x and c_y respectively.

$$\begin{aligned} v_x(0) &= -\alpha \times 0 + c_x = c_x \\ v_y(0) &= -\alpha \times 0 - g \times 0 + c_y = c_y \end{aligned}$$

This gives the net velocity as follows.

$$\vec{v_{net}} = (-\alpha r_x + v_x(0))\,\vec{\hat{x}} + (-\alpha r_y - gt + v_y(0))\,\vec{\hat{y}}$$

Let t^* be the time at which the projectile reaches the peak of its arc. Here, the vertical velocity of the projectile is 0, i.e. $v_y(t^*) = 0$. This lets us get a value for the initial vertical velocity $v_y(0)$.

$$\begin{aligned} v_y(t*) &= -\alpha r_y - gt^* + v_y(0) = 0 \\ \Rightarrow v_y(0) &= \alpha r_y + gt^* \\ \Rightarrow t^* &= \frac{v_y(0) - \alpha r_y}{g} \end{aligned}$$