Implementation

1. **Melee Attack**

To create a melee attack animation, multiple images are prepared in advance, as shown in the example images below.

 A pixelated image of a person on a white moon

Description automatically generated



**Positioning Images**:

A graph of mathematical equations

Description automatically generated  
In Processing, images are drawn using the coordinates of their top-left corner. Since the player is positioned at the screen center (400, 400), the top-left corner of each image must be offset to align the character’s base with this position.

* An offset is stored in a PVector for each image, representing the distance between the base (e.g., the midpoint of the character’s feet) and the top-left corner. For instance, (-15, -44) indicates an offset where the base aligns when the top-left corner is positioned at (400-15, 400-44).

**Determining Direction**:  
The screen is divided into four quadrants using the lines y=xy = xy=x and y=−x+800y = -x + 800y=−x+800. The direction is determined by evaluating the cursor's position relative to these lines.

* A PVector named playerDirection is used to store the direction, where x and y are set to 1 or -1 to represent the player's orientation.

**Animating the Attack**:  
To manage the animation, frameCount is used to determine how long each image is displayed. The frames are cycled through sequentially every two frames, resulting in a smooth animation.

1. **Tower Building**

A Tower class is created, and an ArrayList is used to store all Tower objects.

**Positioning the Tower**:  
When building, the player’s mouse position (mouseX, mouseY) is recorded and passed to the constructor. Similar to melee animations, an offset is calculated so that the top of the tower aligns with the cursor’s position, while the base sits at the mouse coordinates.

**Automatic Attacking**:  
After construction, the tower periodically attacks the nearest monster.

* Using frameCount, attack intervals are regulated by checking whether a minimum frame threshold has been met.
* The position, velocity, and acceleration of bullets are managed using PVector.

A diagram of a circle with a line in the center

Description automatically generated**Calculating the Target**:

For targeting, the tower iterates through the monster ArrayList and calculates the position of each monster:

* The horizontal and vertical distances are calculated based on the monster's position and the tower's base.
* The straight-line distance (RRR) is determined using the dist() function.
* The bullet's speed (rrr) is predefined (e.g., 10). Using the ratio r/Rr/Rr/R, the horizontal and vertical components of velocity (aaa and bbb) are calculated and stored in the bullet’s PVector velocity.

1. **Data Management for Restarting the Game**

When the game ends, starting a new game requires resetting and tracking data accurately.

**Scoring Based on Survival Time**:  
The player's score increases based on how long they survive. To calculate the correct time:

* Use millis() to record the current time. Since millis() cannot be reset, a new variable is introduced to store the starting time of each game.
* Subtract the start time from the current time at game over to calculate the elapsed time.

**Resetting Game State**:

* All ArrayList objects, such as bullet and monster lists, are reinitialized as empty to clear previous game data.
* Other game elements, such as player health and materials, are also reset to their default values.