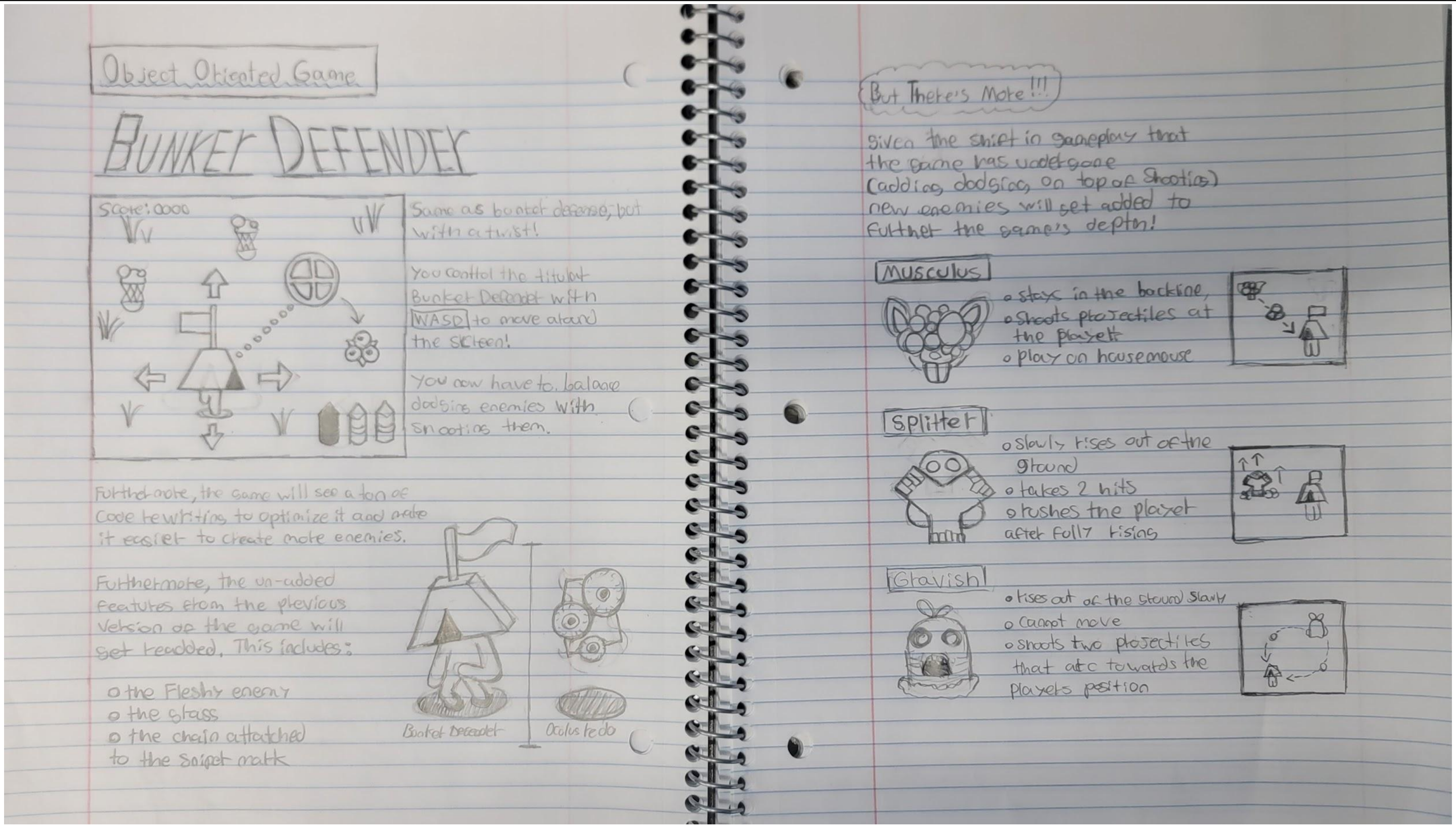
Bunker, Defender

Use this to summarize your idea, plan it using sketches, notes and pseudocode as needed

My idea is to take Bunker Defense (My Object Oriented Toy) and expand upon it greatly by introducing a twist; You now control a fully mobile player-controlled bunker. This way, you now not only have to kill the incoming enemies, but you also have to dodge them. This will also be a great opportunity to revisit unimplemented enemies, and create new ones to add further variety to the game. Lastly, I also plan to heavily rework and optimize the code I pull from the original Bunker Defense to make it easier to work with, run faster, and overall read better

Sketches of Game Idea:



Object Oriented Game

Bunker Defender

Score: 0000

Same as bunker defense, but with a twist!

You control the titular Bunker Defender with WASD to move around the screen!

You now have to balance dodging enemies with shooting them.

Furthermore, the game will see a ton of code rewriting to optimize it and make it easier to create more enemies.

Furthermore, the un-added features from the previous version of the game will get readded. This includes:

- the Fleshy enemy
- the grass
- the chain attached to the sniper mark

Bunker Defender

Oculus redo

But There's More!!!

given the shift in gameplay that the game has undergone (adding dodging on top of shooting) new enemies will get added to further the game's depth!

Musculus

- stays in the backline,
- shoots projectiles at the player
- play on housemouse

Splitter

- slowly rises out of the ground
- takes 2 hits
- rushes the player after fully rising

Gravish

- rises out of the ground slowly
- cannot move
- shoots two projectiles that arc towards the players position

Pixel Paper Layouts:
(Yes, layouts! I realized last time I actually never made a pixel paper page for the game over screen! This is now included with the other usual pixel paper, and as such, is significantly more polished!!)

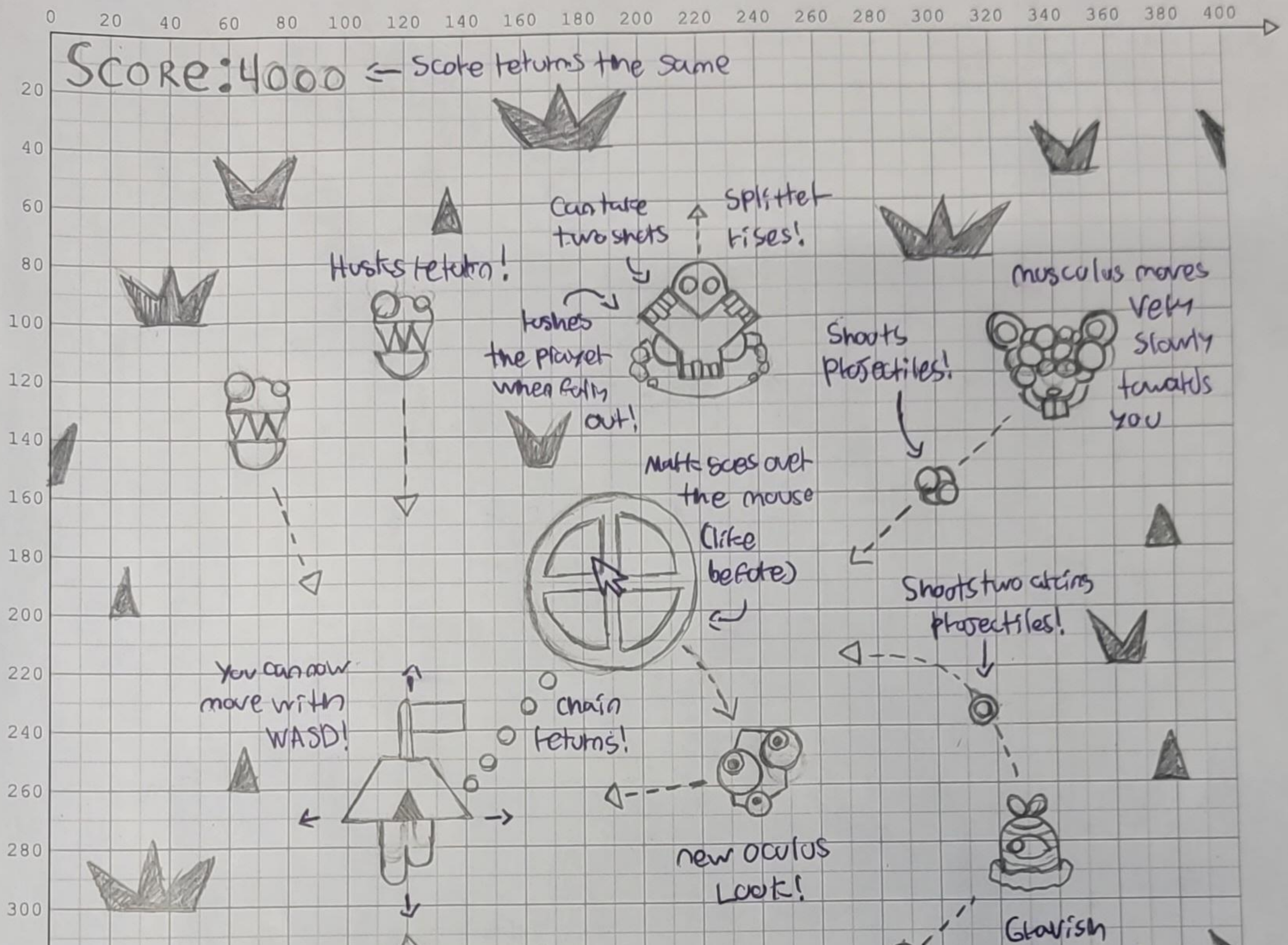
Legend:

⋰ dotted lines for movement

╲ hard lines for drawn elements

╲ purple for notes

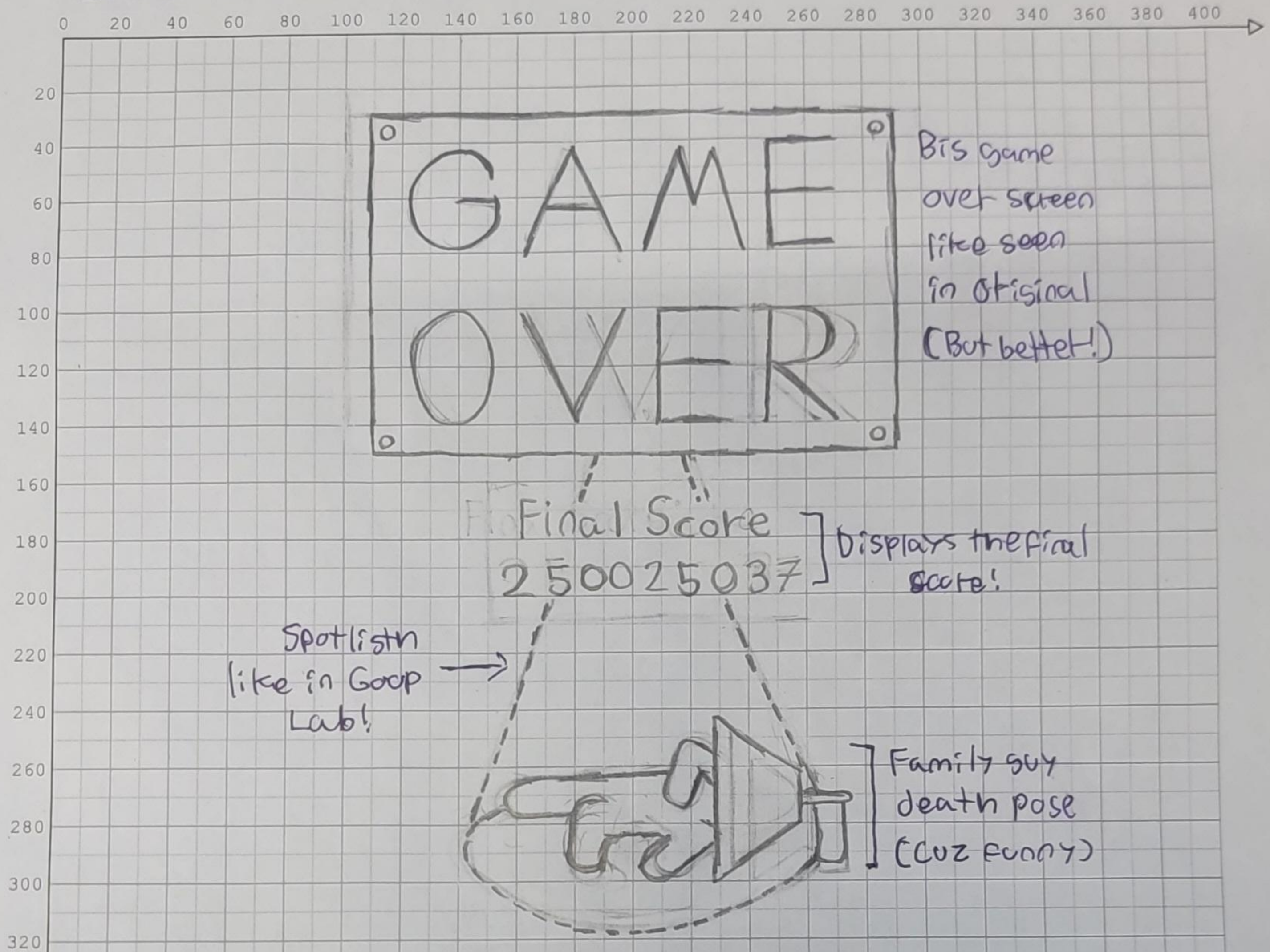
Main Game Screen


0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
Score:4000 ⇐ Score returns the same
Husks return!
Can take two shots
Splitter rises!
Rushes the player when falling out!
Shoots projectiles!
Musculus moves very slowly towards you
Matt goes over the mouse (like before)
Shoots two arcing projectiles!
You can now move with WASD!
chain returns!
new oculus look!
Glavish

Legend:

- dotted lines for transparent elements
- hard lines for drawn elements
- purple for notes

Game Over Screen



GAME OVER
Big game over screen like seen in original (But better!)
Final Score
250025037
Displays the final score!
Spotlight like in Goop Lab!
Family guy death pose (cuz funny)

Pseudocode:
Similar to the original's Pseudocode, with some key changes to mirror the goal of optimizing the project

bunkerDefender (Main)

Setup

Set up ints that control the number of bullets, the score number, the x and y coordinates of the player, etc.

Set up objects, such as the enemy class, the enemy subclasses, the player, and the grass objects
Put the enemy lists in their own larger list for easier access (This will be called enemyTypeList)

Set up the default draw settings (TextMode, rectMode, noStroke, strokeSize, etc).

Set up a gameState string (set to "mainGame" by default)
Set Up a Switch Statement that uses the gameState string to determine when certain things should happen

For the grass list:
Set up a random x and y location for each grass piece in the list as well as a randomized grass type for every grass in the list.

Draw

Case "mainGame" draw:

Draw the main game background
For the length of the grass list:
Draw the player (bunker defender) on the player Pvector.
Draw the grass (Function that draws green-ish triangle bunches)

For the length of the enemyTypeList;

```
For each different enemy list within the larger list (a loop within a loop!)
Draw enemy in each individual list.

Draw the sniper sight (function)
(Draws sniper sight based on a pVector that updates its x and y based on the mouse’s x and y)
(Draws dotted line from the player’s vector to this mark vector using circles drawn along the vector)

For the base number of bullets (aka, the number of bullets you start with)
        Draw empty bullets (which remain for the rest of the game and are used to tell the player when they’ve used a bullet)
For the number of bullets,
        draw a bullet (function that draws the rest of bullet ui)

Draw the score text (using score to update it)

Case “gameOver” draw:

Draw the game Over background
Draw the game over box and the game over text on it
Draw the replay button and the “Continue?” Text on it
Draw the Final score text (Based on score at time of death
Draw the “Dead Player”
Draw the spotlight
```

Functionality

```
For the length of the enemyTypeList;
For each different enemy list within the larger list (a loop within a loop!)
Call the enemy’s movement function
Call the enemy’s shoot function (Inherited as empty by default)
Call the enemy’s “Has reached center” function
(If they have, switch to game over screen)
```

```
If the W, A, S, D keys are pressed
Call the player objects' movement function with the key character as input.

Overrides the mouseClick prebuilt function to do this instantly (as per suggested in the object oriented toy feedback):
(Otherwise, works exactly the same)
    If the player has bullets
    If the delay between shots is done (This delay is seen in the original to prevent running out of bullet instantaneously)
    Reduce the number of bullets by one
    For every enemy list:
        Has the mouse been clicked over the enemy?
        Are they out of health?
        If yes, enemy is sent to a random x and y position off screen
Else (there are no bullets);
    Increase bullet reload time by one
If bullet reload time is equal to or greater than 60 (1 second)
    Reset bullet number back to 3

If the user clicks the "Continue?" Button:
Set the screen to a "setup" screen.
Reset all relevant values back to their default values (example: score to 0, player position to center, randomized enemy positions)
```

Objects

Enemy Class

This time around, I will be using inheritance for this project, creating a main enemy class with all the required functions that can be overridden to determine the enemy's specific behavior. The functions include:

A function for when an enemy has been shot
A function that checks if the enemy has reached the player's position

An empty draw function
An empty enemy movement function
An empty projectile shoot function.
A function that creates a directional vector between the player position and the input position

As such, many of the variables set up for the enemies will minimize the need of custom numbers as much as possible
By that, I mean that there will be a *lot* more enemy variables
Every enemy inherits from this class, and their behaviors are overridden accordingly. These enemies are:

Husk: Basic enemy, moves towards the player and gets slightly faster as it does so
Oculus: Moves towards the player, but away from the mouse's position
Fleshy: Moves in a grid pattern towards the player. Has 2 health!
Splitter: Rises from a random spot in the ground for a while, and once fully out, rushes the player at monstrous speeds
Musculos: Moves veeeeeeeeery slowly towards the player, but shoots projectiles towards their last position!
Gravish: Doesn't move, and instead shoots two projectiles that arc towards the player from the sides!

Projectile

Similar to the previous enemy inheritance, this too is a class which the two in-game projectiles inherit from. As such, in the same way, it has a variety of functions that deal with movement and drawing the projectiles themselves. Nothing too disimilar from the enemies, except that they do not take damage.

Player

Setup necessary player variables
Movement is based on the key pressed, and uses a switch to determine which direction the player character should move towards
Draws the player character (who's feet move as they move!)

Grass

Draws the grass based on their x and y pVector coordinates.
An additional "type" variable determines the type of grass that is drawn, which varies between:

3 Triangles spread out in a larger pattern resembling a crown
2 Triangle put against each other resembling a small mouth
1 single taller triangle

Where will the inventory skills be demonstrated? List every one to be sure you've included them.

Shapes:

1. Will show in grass, enemies, etc
2. Stroke used in new enemies
3. Mode changes used regularly for enemies and in the game over screen text

System:

4. Setup/draw in main screen
5. Background for new game over screen, random used regularly in enemies
6. Constraint used to prevent player character from leaving the screen
7. MousePressed() used for new shooting, keyPressed used for player movement
8. Increment operators used in new enemy class for when the player's location is the same as the enemies location as well as in most new if statements present (Example, movement for player)
9. New local variables created for the new bigger enemy list for loop within for loop
10. Global variable in the form of the new integers/strings such as gameState and playerX and Y

Debugging:

11. Print used specifically to figure out what in the world is going wrong with fleshy

Control Flow:

12. New if statements for the player movement.
13. New use of booleans in the main enemy class.
14. Logical operators used for bullet shooting rework (as well as constraint code for player movement)
15. Switch statement for game screen switching, as well as in-object player movement

Loops:

16. New for loop for setting up new grass object locations and types
17. New for loop within for loop used to select enemies within the list of enemy lists.
18. A break() statement for most switch statements to prevent them from moving onto the next case accidentally.

19. A while loop does something as long as a specific condition it is given remains true, akin to how an if statement works. A for loop, on the other hand, uses a local variable, a condition, and an incremental of this local variable to loop a specific amount of times based on all of these things.

Functions:

20. Function that draws the “dead player” on the game over screen
21. The new “has touched player” function in the enemy class the other enemies inherit from which returns a true or false when called
22. Parameters in a function are the input materials the function requires listed inside the brackets of the function, while an argument is the information provided to this function to meet these very requirements when it is called. An example of this relationship; a function may have a parameter int to churn out a calculation, and as such, when it is called, it must be provided this integer as an argument.
23. The main directional vector function will take a parameter in the form of the player vector. Similarly, the x and y coordinates of the player are parameters for the previously mentioned “has touched player” function.
24. The function that shoots projectiles requires a projectile object, and as such, will be set as one of its parameters

Classes/Objects:

25. A class is the blueprint that the object follows when it is created. For example, when a husk object is created in my game, it uses the husk class as its blueprint for all its functionality
26. A constructor function takes arguments and uses them to set the specific variables of an object created using the class. It does this when an object is initialized by using an overridden default constructor within the class itself, which as its name suggests, sets the default variables of the object
27. It's for ease of use and access. Having classes as different tabs within processing also allows us to work on multiple classes simultaneously, and makes it easier to see the big picture of how the classes communicate between each other and main.
28. The game primarily uses classes with constructor functions. In particular, it uses them for the player, the enemies, the projectiles, and the grass.
29. The keyword new is used whenever adding more objects into the different object array lists by using for loops.
30. The player uses parameters in the form of their default x-and-y coordinates. Similarly, this is also done for all the enemy classes.

Lists:

31. Arrays, unlike arrayLists, have a significantly higher amount of constraints attached to them. For example; you can add new ints to an int arrayList, but cannot do the same for an Array of ints as its length is locked.
32. If you don’t know the length of a given list (say, it is inputted by the user of the program) it is perhaps easier to loop through the list backwards by using the length() command and decreasing the length() number by a greater and greater value. This is specially relevant if the size of this list suddenly changes, as now because you start at the end of the list, you do not run the risk of going over the lists’ length.

33. The list of enemy lists will come in the form of an array, as the lists within it can change, but the array itself shouldn’t have to after it is initialized
34. The new enemy lists for the splitter, musculus, and gravish are all ArrayLists like the previous enemy lists in the original project
35. As previously mentioned, the enemies are managed within an arrayList. Another example of this is the grass objects, which are also managed within an ArrayList
36. Size() will be used to determined how many times the nested for loop should loop through a specific enemy list

Vectors:

37. Pvectors become especially useful when dealing with more complex movements. Pvectors have velocity and acceleration built into them allowing for significantly easier changes in movement that would otherwise require writing velocity and acceleration formulas yourself. Furthermore, it also allows for much more complex movement, such as, in my game, an enemy moving towards a specific point while simultaneously trying to avoid another using directional vectors.
38. Pvectors are used to determine the movement of enemies based on their own Pvector and the Player Object’s
39. Fleshy, like the other enemies, will use acceleration. However, Fleshy’s movement stops suddenly, before shifting direction and accelerating rapidly again, giving his movement significantly more weight and gravity.
40. Directional vectors are used to accurately direct the movement of the enemies towards the player regardless of the players’ location. This is done with a really complex calculation, which figures out the next position the enemy should be at based on its position and the player’s.
41. The locations of the enemies are all randomized on program start, and change whenever they are killed to be randomized once more.
42. A normalized vector, also known as a unit vector, is a vector whose magnitude (length) is equal to 1, focusing on the vector's direction without the need to account for its length.
43. The copy() command will become extremely useful when creating projectiles, as they do not track the player, and instead move to their last position. Likewise, Fleshy’s movement being grid-like should greatly benefit from only taking the position of the player occasionally (once again, using copy()).

Optional:

44. Timers are used to delay the bullet shooting as well as adding reload delay. Furthermore, timers will be used for the movement of Fleshy and Splitter, as well as delaying the shooting of Musculus and Gravish
45.
46. Buttons were part of the original project already, and will get reworked.
47.
48.
49. Enemies detect when their vector collides within a certain radius of the player
50.

Milestone 1	**Milestone 2**	**Milestone 3**	**Milestone 4**
I will deliver a more optimized version of the original bunker Defense object oriented toy. This more optimized version will include object inheritance for enemies, switch statements, better enemy list looping/storing, mouseClick function overriding, and a better button function. In terms of new things, it will add player movement using keys as well as a new player avatar to go alongside this change. Furthermore, the grass originally cut from the game will return as an object and slightly altered to be more plausible	I will deliver a new set of enemies to the game, including an old cut enemy, and then introduce a new mechanic to the game: Projectiles!. First, I will reintroduce Fleshy to the game, with all of its original functionality. Then, I will create two new enemies afterwards: Musculus and Splitter. Of note is Musculus, who will use the new projectile object class to attack the player from afar.	I will add one more enemy, as well as polish the visuals of the game further from their original form. The last enemy, Gravish, does not move, but instead shoots projectiles which move in a parabola arc towards the player's last location The end screen will see a visual revamp to look much nicer, and so will too any older assets from the original iteration of the game	No particular plans past this. Perhaps an extra enemy if possible. I will be largely focusing on further polish at this point Essentially, this is my buffer in case things go very, very, veeeery wrong.
Which inventory skills will this demonstrate? List them.			
1	11	20	
2	24		
3	34		
4	38		
5	39		
6	40		

7	41		
8	43		
9	44		
10	49		
12, 13, 14, 15, 16, 17, 18, 21, 23, 28, 29, 30, 33, 35, 36, 46,			
You should deliver approx. 10 skills at this milestone	You should deliver approx. 10 skills at this milestone	**You must deliver 30 inventory skills by this milestone.**	