Squadron

Author : Vivek Bigelow

About - Squadron is a clone of the Team17 game Worms built using the pygame engine. Worms is a 2D turn-based strategy artillery game where a player(s) control a team of worms and try to eliminate all the worms on enemy teams. I chose to replicate Worms because there are several interesting features that make up the game. These features include:
Randomly genereated destructible terrain, A.I.,a basic physics engine, turn based strategy, inventory, different weapons and items with different behaviors,
multiplayer capabilities, etc.

Implemented Classes:

Terrain - Object that stores the noisemap matrix,bitmap matrix, and pygame.Surface object of the terrain.

member variables:

screen - the pygame.display that the game is drawn on. This is used to get the dimensions of the terrain.

x - the width of the game screen, used to create the dimensions of various matricies

y - the height of the game screen, used to create the dimensions of various matricies

noisemap - 2d numpy array generated by the perlin noise function. Used to create the black and white bitmap

bitmap - pygame.Pixelarray of pygame.Color(255,255,255) or pygame.Color(0,0,0) objects.

surface - pygame.Surface generated by the bitmap PixelArray. Needed to draw the bitmap onto the screen.

methods:

perform_convolutions(self) - method that performs 4 largeblurs and one sharpen on the noisemap to blend the perlin noise into a more natural shape. Utilizes the scipy.signal.convolve2d(matrix,kernel,flag) function.

add_darkness(self, noisemap) - method that adjusts the randomly generated noisemap to be darker on the sides and top to create a more natural island.

add_light(self,noisemap) - method that adjusts the randomly generated noisemap to have more solid sections.

generate_noisemap(self) - creates the 2d numpy array from the perlin noise function and then uses perform_convolutions() and add_light() and add_darkness() to make the terrain more natural.

generate_bitmap(self) - create pygame.PixelArray from the noisemap where points in the noisemap that are less than 1 become pygame.Color(255,255,255).

PhysicsObject - Object that stores the physics values for any object in the game that responds to physics.

member variables:

px - position on the x axis
py - position on the y axis

vx - velocity in the x direction
vy - velocity in the y direction

ax - acceleration in the x direction
ay - acceleration in the y direction

radius - radius of the bounding circle
friction - friction the object receives when responding to a collision.

Soldier - Object that extends the PhysicsObject class and pygame.Sprite class. The Soldier is the basic unit that the player controls during their turn.

member variables:

image - pygame.image loaded from the load_png function.

rect - pygame.rect object, obtained from loading the image.

terrain - The terrain of the game. Used for collisions.

methods:

update() - Function that updates the motion of the soldier. Also checks for collisions. First the method calculates the new physics vectors and finds the potential position. The method finds the angle of motion and then iterates over points that cover the half of the bounding circle in the direction of motion. The points are close enough that a pixel can not fit between them. Then the method calculates the response vector and response velocity if there is a collision, if not the object continues in its direction of motion. If there is a collision then the object moves in the opposite direction of impact and is slowed down by friction.

Boom - Object that represents an explosion in the game. Updates the bitmap where the explosion took place and determines which objects are hit by the explosion.

member variables:

terrain - The games terrain object. Used to manipulate the bitmap for destructible terrain.

sprites - a list of all of the physics objects in the game (probably should be renamed). Used to update the physics vectors of the object if it is in the blast radius of the boom

worldX - x position that the boom occurs at. Center of the circle.

worldY - y position that the boom occurs at. Center of the circle.

radius - Radius of the explosion.

methods:

drawline(self,sx,ex,ny) - Takes start of x and end of x and draws a line from sx to ex in the ny row. The line is drawn in the bitmap pixelarray of the terrain.

circle_bresenham(self,xc,yc,r)- Takes the center x and y coordinates and the radius of the circle to be drawn. Uses 4 calls to the drawline function to draw a series of lines that will make up a circle.

shockwave(self) - Iterates through all of the physic objects in the game and determines if they are in the blast radius of the boom. If they are then the velocity vectors are changed to react to the blast.

main() -

Objects:

screen - pygame.display
background - pygame.Surface
terrain - Instance of the Terrain class
sprites [] = list for storing each physics object
allsprites - pygame.sprite.Group that will hold the sprites
clock - pygame.Clock

Game Loop:

Framerate = 30

When B is pressed an instance of the Boom class is created at the mouse position.

If the mouse is clicked an instance of the soldier class is created at the mouse position. The soldier sprite is added to allsprites and appended to the sprite list.

allsprites is clear,updated, and drawn

The terrain is updated for destructions

Original Unimplemented Game Features:

Teams: In the original game, players are able to name their team and the worms on that team. Teams can have a different number of worms, but the more teams in a match the fewer worms there can be on a team. Attacks from a worm can hurt everyone in the game, including the attacking worm and its teammates.

A.I.: The original game implemented an A.I. with a sliding scale of difficulty. The better A.I. was better at aiming and eliminating player controlled Worms. The A.I. needed pathfinding, inventory control, enemy locations, and general strategy. Each A.I. worm might not need to "interact" with each other but they would need to know not to attack each other.

Inventory: In the original game each worm has an inventory of different items and weapons. Some items are fo moving around like a parachute and bungee cord. Other items include walls for defense. Most of the items are weapons for combat. The main weapon is the bazooka that has unlimited ammo. Other weapons include grenades, mines, homing missiles, airstrikes, and holy hand grenade.

Turn Based Combat:Each player can control one worm per turn. The turns are the "main" state machine of a game. A turn can end when the player commits an "action" with a worm, the worm is injured or dies during the turn, or the time limit of the turn runs out.

User Control: The player controls one worm per turn and the worms can move left or right and jump. There is a limit to the height of wall that the worm can climb up. The player uses one worm to commit an "action"where they select one item and use it to complete the turn.

Wind: In the game players use weapons that use basic projectile physics. The projectiles are influenced by a wind force vector that randomly changes through the game. The wind makes it harder for players to aim and accurately hit enemy players.

Extended Features: When thinking of the game there were some features that I think would be fun to add to the original game.

Soldier Classes - In the original game every Worm has the same behavior. A way to add more strategy to the game would be to have soldiers with different behaviors and items. For example a medic class that is used to heal soldiers on your team. Or a scout class that can move farther than other soldiers in a turn, but is unable to take lots of damage.

Saveable Teams - In the original game players are able to name their team and the Worms on that team. I think it would be great if a player could save teams they create and the game would store data on that team and the soldiers on it, for instance things like number of mathces and victories.

Level Trees for Classes- If Teams can be saved then it is possible to implement a experience system and levels on the soldiers and the different classes. This can work something like Pokemon and other rpg games where the soldiers acquire new skills and items as they level up.

Network Multiplayer - If a player's team is getting better then eventually they may want to play against other teams and players. The game would benefit from a networked multiplayer system.

References and Sources:

External Libraries and Packages:

pygame - The game is built using the pygame engine.

math - used for sqrt(),atan2(),cos(),sin(), and math.pi.

os - Used for getting the path to images in the asset folder.

numpy - I use numpy matricies for several data structures in the game. I also use the numpy.arange() method to iterate through a serires of floating point numbers when checking for a collision on
an object.

scipy.signal.convolve2d()- Method that takes a 2d array, a convolution kernel, and a flag. The method returns a 2d matrix that is the result of the convolution. The flag dictates whether the resulting matrix is the same size as the original.

perlin2d.generate_perlin_noise_2d() - Method created by that generates a 2d numpy matrix with filled with values from -1 to 1 that have been calculated the perlin noise function. I use this function to generate the original noisemap matrix which is a member of the Terrain class.
Blog: https://pvigier.github.io/2018/06/13/perlin-noise-numpy.html
Github: https://github.com/pvigier/perlin-numpy

Terrain Generation:
References:
Red Blob Games - Making Maps with noise functions
Link: https://www.redblobgames.com/maps/terrain-from-noise/

Game Development Stack Exchange Example

https://gamedev.stackexchange.com/questions/20588/how-can-i-generate-worms-style-terrain

Explanation and Code Link: https://juliango202.com/terrainver/?terrain=eyJ3IjoxMzgwLCJzZWVkIjowLjExOTA3MTk1ODQyNDMyNDcyLCJub2IzZSI6MzUsInR5cGUiOiJ0eXBlLTMiLCJjaGFyYXMiOjl9

Convolution Matricies:
Machine Learning Guru Post on Image Filtering
http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

Collisions and Destructible Terrain:
This is where I learned how to achieve collisions with the white terrain and how to use the Circle Bresenham function.

One Lone Coder- Code it yourself Worms

YouTube: https://www.youtube.com/watch?v=EHlaJvQpW3U
Github Code:https://github.com/OneLoneCoder/videos/tree/master/worms