**Game Overview**

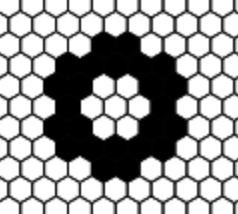
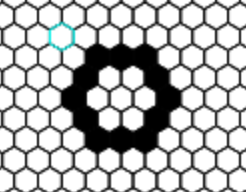
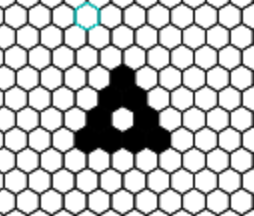
Morph is 2d MMO shooter written in JavaScript, with an AI backend written in python. The player begins as a small ship on a large map. As the player destroys space-debris (planets, asteroids, etc..) around the map, they will accumulate points to add features to their player. The game may be broken into five distinct objects: players, projectiles, AI-bombs, space-debris and gravity traps.

**Players** have the ability to move in two dimensions throughout the map. A player may shoot either projectiles or AI-bombs at other players or space-debris to accumulate points, which add features to their player. Different upgrades are available to the user including: health, number of guns, bullet projectile speed, bullet reload speed, bullet size, AI-bomb projectile speed, AI-bomb spread, AI-bomb explosion size.

**Projectiles** are expended by the user at other players or space-debris. When a bullet comes in contact with other objects on the screen, a collision occurs. Once a collision occurs, the health of an object is diminished in proportion to the health of the counter-object.

**AI-bombs** are generated by the user, but their explosion pattern is determined by a genetic algorithm. The algorithm takes in a two-dimensional grid of the space surrounding a selection point. It then uses this grid to test the “fitness” of an explosion pattern. Using the theories behind Conway’s *Game of Life*, we generate different gridded starting positions of our “bomb” as shown below:

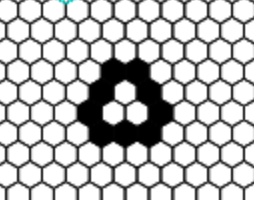
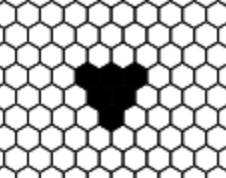
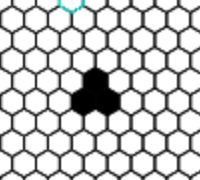
**Position 1**



2

1

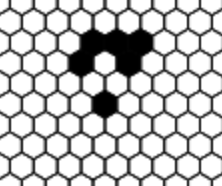
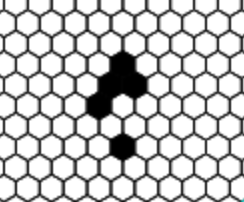
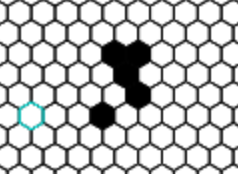
**Position 2**



2

1

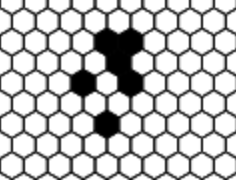
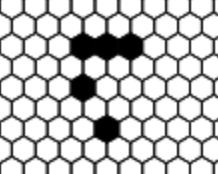
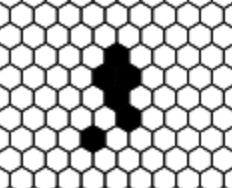
**Position 3**



2

1

**Mate 2/3**

****

2

1

Each starting pattern determines the following configuration and the result after N number of generations will allow us to test the efficacy of the starting arrangement. If an explosion hits or comes close to its target, it is awarded a fitness score proportional to its determined efficacy. The higher the fitness score, the more likely this starting position will position itself into later generations.

**Space-debris** float around the game map and are primarily available so that users may gather points sans interaction with other players.

**Gravity traps** float similarly to space-debris around the map but may be used offensively or defensively by the player. A player may shoot its projectiles into the trap. Once the projectiles enter the gravity trap, their trajectory is dictated by the shape of the trap itself. The player with the most projectiles in a trap “owns” the trap. The owner may either use this trap defensively or release all their projectiles from the trap. Upon release of the gravity trap, the trap will “die” and all projectiles will continue in their trajectory vector at their time of release.

**Work Completed to Date**

Gameplay - Stephen

**Overview**

*Gameplay -* The gameplay basics have been setup thus far. A player is able to control themselves on the map and shoot projectiles. As the players health diminishes from contact with projectiles, their player may die and respawn in a new location. The basics of the AI-bomb have been setup from a display perspective. The logic of successive generations has been implemented, but AI bombs are not properly implemented in the game. They still need to be integrated into the python AI and have appropriate attributes such as health. Gravity traps have not been integrated in the current version of the code, but the logic was demonstrated and generated in previous code. Their current state will need to be modified before integration.

**Issues**

*Flickering canvas* – I couldn’t figure out why the canvas was flickering at random times in the game. The objects seem to be drawn at intervals not stated in my code. I had originally set the redraw to 60fps, but realized that the refresh rate of the actual browser window was causing problems. Instead, I kept my variables updating at 60 times per second, but set the redraw to be parallel to the window refresh.

*Class Abstraction –* Classes have been a huge nightmare up until recently. Their implementation is rather new in JavaScript so experimental settings have to be set in Node.js. ­ There are also some behind the scenes issues sending classes through Socket.io in which class functions are not able to be sent. Only variables cross the server line so any class functions need to be implemented where they are used. Setting up my game in a way that logically makes sense was also quite difficult. I realized after much trial and error that I am able to abstract my game from the server logic and make simple calls the game class such as Game.start() and Game.add\_new\_player(id).

*Screen Orientation –* This is something that has taken quite a bit of getting used to. HTML canvas is very unique in its implementation. I’m not sure I can explain what I’ve learned thus far, but getting used to orienting objects and the game screen is continuously challenging.