**Data-Oriented Technology Stack (DOTS)**

Counter Gambit is made primarily using Unity DOTS in order to test out the experimental system and its performance benefits. DOTS uses parallel processing, their burst compiler, and clever data-oriented programming techniques to optimize performance.

**Enemies**

The primary test of the benefits of DOTS is our ghosts. With DOTS we should be able to have lots of enemies at once. Enemies are entirely physically simulated and use purely forces to path to the player and avoid obstacles.



**Networking**

Counter Gambit was initially planned to be a multiplayer game using GGPO rollback networking, and utilizing the control over the update loop DOTS provides. However, after

investing a regretful amount of time into its implementation, we discovered that there is

an incompatibility with key DOTS packages and serialization of data(which is required for networking).

**Shop**

Players acquire items by purchasing them from the shop. The shop is opened at the end of every round, and always starts with an allotted amount of items from each rarity. These items are also randomized everytime the shop is opened. After buying an item, the slot is replaced by an item of the next rarity. 

**Items**

Passive items have made it into the game with Active items planned but not yet implemented. These passive items each have unique effects which can stack indefinitely by purchasing duplicate items.



**Parallel Processing**

Any processes which can affect many of the enemies (such as damage over time effects) are scheduled using DOTS’ job system and support for parallel processing.

**Systems, Components, & Entities**

DOTS operates by creating containers for generic data containers (entities) and grouping them together based on the type of operations you want to perform on the data (components). The operations (systems) are specific, and must contain a query for the entities it wants to operate upon where the data in the entities is the search parameter. E.g. if you have many different types of animal objects but they all have the same class of health data component, DOTS will represent each animal as a container for the visuals of the animal and the health data, it will group them together in memory as best it can based off the shared data (health), and perform all health calculations at once thereby optimizing performance.

**Collision Filtering**

When handling physics body collisions DOTS has built in controls for fine tuning which collisions to check, saving processing time as well as code creation and testing time.

**Integration**

DOTS can integrate with existing Unity code and classes with relative ease - specifically the UI and camera aspects.