# Secret Feature for RoadRage

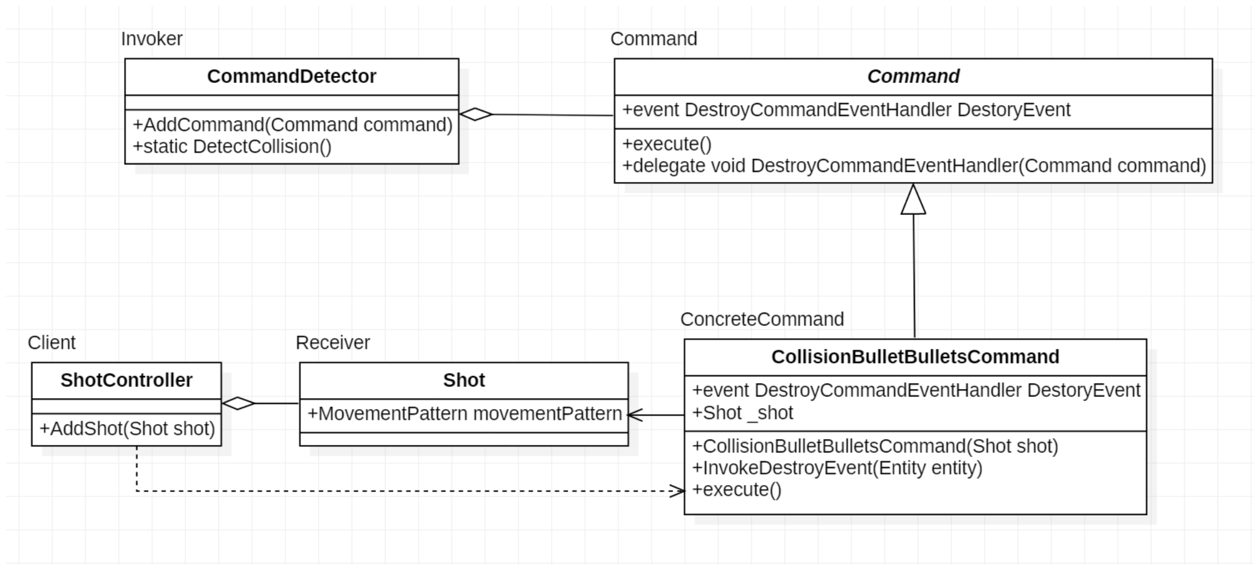
Team: Code Crusaders
Project: Road Rage
Team Members:
Connor Strom
Emma Fletcher
Logan Kloft
Manjesh Puram

I. Target Feature: **(B)** Collision detection among different bullets

We are going to detect collisions between shots. We'll use collision detection to create 'sticky' shots. Sticky shots change the movement pattern of any shot they collide with to their own movement pattern. We believe this behavior best mimics the behavior demonstrated in the video for secret feature B.

II. Design Plan:

We plan to extend our current Command Pattern which handles hit detection. Here is the class diagram of the command design pattern for collision detection among different bullets only.


Invoker
CommandDetector
+AddCommand(Command command)
+static DetectCollision()
Command
Command
+event DestroyCommandEventHandler DestoryEvent
+execute()
+delegate void DestroyCommandEventHandler(Command command)
Client
ShotController
+AddShot(Shot shot)
Receiver
Shot
+MovementPattern movementPattern
ConcreteCommand
CollisionBulletBulletsCommand
+event DestroyCommandEventHandler DestoryEvent
+Shot _shot
+CollisionBulletBulletsCommand(Shot shot)
+InvokeDestroyEvent(Entity entity)
+execute()

The CommandDetector stores commands and calls their execute function every frame to detect collision, which is why it's labeled as the invoker. Whenever a 'sticky' Shot is added to the ShotController, it creates a new CollisionBulletBulletsCommand and adds the command to the CommandDetector. Thus it acts as the client in the command design pattern. The ShotController also contains all shots which is why there's an aggregation arrow instead of simply a directional relational arrow like the client normally has to the receiver. When a collision is detected in the execute() function of the ConcreteCommand, it changes _shot's MovementPattern to the MovementPattern of the shot it collided with, but only if the collided with shot is not also sticky. If a sticky shot collides with another sticky shot, both shots maintain their own movement patterns.

## III. Design Impacts:

Collision detection will also trigger on shots created by the player. In the case of sticky shots, player shots will be affected. If we don’t want this to occur, we might include an extra attribute on each shot which specifies the source of the shot. For example, “player” or “enemy”. In the collision handler of each shot, we can add an ‘if’ statement to check whether the source of the shot is a player, and if so handle collision differently such as ignoring it entirely. Although, it can also be argued that sticky shots behaving this way is both logical and a desirable feature.

Since we are iterating over the entire list of shots for every shot, there might be noticeable frame drops. The time complexity of shot-on-shot hit detection is $O(n^2)$ where n is the number of shots in the ShotController. As you can imagine, in a bullet hell game there might be many shots at any given time. One suggestion for improving hit detection is only checking objects in a certain area around the bullet. Some potential data structures are discussed in this article and many forums seem to suggest using a Quadtree.

For creating the sticky shot behavior, we’ll need to extend the current ShotParams class to accept an extra boolean field called “sticky”, which also means the DeepCopy function will need to be updated. And then in our json file, we’ll have to add the “sticky” attribute and give it a value of “true” for shots to have sticky behavior.