Hui Gao

CS108

Vooga

Fighting Game Camera: A Design Proposal

One interesting problem that we will need to face in our fighting game is how the camera will work. There can be many orientations, movements, and configurations. For example, Street Fighter type games have a side to side scrolling camera that does not allow the characters to move off the screen at all (with a background that follows in the scrolling), whereas a game like Super Smash Brothers will balance the camera out to capture everyone on the screen as well as the terrain by zooming in and out, and panning. Since we want the developers to be able to pick which kind of camera they want to use in their game, as well as allow the users to have the option to change these settings right before a game or when they create a level, and also allow different combinations of panning, zooming, scrolling, etc., an elegant design will be more complicated than having a simple config file.

The first aspect we should consider is being able to have different combinations of camera techniques that will allow developers to distinguish a SSB-type camera from a SF- or KOF-type camera. There are 8 big types of camera types, fixed point, rotating, scrolling, movable, floating, tracking, pushable, and first person. Some of these such as first person will not be relevant but most of them will be. One design pattern we could use to make this a better design is to use a decorator pattern, where we can have a simple default camera class that simply shows the entire level and doesn't move, and decorate it with zoom or scroll functionality as necessary at runtime (which can be accomplished using strategy method). This will potentially allow us to have many different camera features for an array of levels that can be decided at runtime. We would have to make sure that certain camera types do not conflict, for example a fixed point and scrolling camera.

Although each level will have a predetermined camera configuration, the developer should be able to choose to allow the player to change the configurations right before playing a certain level. In order to accommodate this, we could just decorate a camera based on the level's configurations at runtime, and then continue wrapping it with decorations if a user makes a change to the camera configurations. In this case we would need to account for conflicts between the player's camera mode and the level's default camera mode. We would want to override any conflicts in the default configuration with the user's choice. With decorator, since the outermost occurrence of a method is used (i.e. zoom() or scroll()), if we decorate the camera with the defaults and then the user choice, the camera configurations will be the desired ones.

I feel that this problem will be challenging to implement with this proposed design solution. Not only do we need to consider the different combinations of configurations possible, but how they interact with each other as well. The behavior of each camera mode will need to be able to do this to give the desired camera style.