Survive In 60” Game Analysis

Time Review

This coursework took average 5-6 hours in a day and five days to complete the goals specified in the plan. The first step was to understand the structure, functions and some examples of JGame, which took a day time. Then, the project went to the step of building the structure and completing the details within the structure. But after debugging the first version of my game, it seems infeasible to add new features (i.e random movement of star and invisible function of player)
so that I refactored some classes and put them as inner class of main. This design really help to get rid of the complicate reference transfer in the case I used separate class file of player and star. As inner class, player and star can directly use the attributions of main engine. Graph and media designed was re-polished in the last day.

Readability

Like several samples do in JGame, my game has inner classes. Although this may effects the readability. But this style of code can solve the back channeling problem, because inner class can directly use the attribution and method of the main class rather than use reference or parameter of method. To be an elegant code, the game still need to be polished by building the hierarchy of level of game. Extracting the three level into separate class can make the main class more concise and elegant.

Extensibility

My code is extensible to add new elements such as new types of enemy or power-ups. In this case, the new classes only include overwritten methods: move, hit and hit_bg and add the one line of collision control code in the doFrameInGame method of main class. However, my game have much workload to modify the process of game such as adding an inferno mode after difficulty mode, in which all process after difficulty mode have to be modified and some methods in inner class have to be changed. In such case, my game do a poor job in extensibility.

Testability

My game were divided into three parts for testing. First, only keep the background process to run. Then add the star element for testing the movement and collision with boundary. Finally, add the player element for testing the movement of player, collision with star and skills of player.

The movement of star and player were easily to test since all the movement rail can be identified by the eye. But the process after collision were hard to test. That’s because a collision cause the state jumping into another state. To guarantee the whole process working well after jump is a little more torching.

Conclusions

This game reach the expectation of the gaming. But to the coding aspect, the game cost too much lines of code to deal with the different situations in different levels, in which I normally use if-else rather than use concise structure. To this point, I didn’t do a good job. This problem can be avoid by extracting the levels into new classes.

Future work

I would like to implement the interface for more extensibility. For example, create an interface for level so that different level should apply the some structure, which is more easily to add a new level without modifying other structure. This interface include the method for generating the star, and movement of star.

Good code design

This code is from star class which control the movement of star when initialized. Star is initialized in the off-screen boundary so that we can see the star flowing from the outside boundary into inside of boundary. After across screen boundary, the star can only run with the in-screen boundary.

To guarantee all stars flow into screen rather than disappear in off-screen. I design the bounce function for off-screen boundary. When the star enter or bounce into the screen, the screen boundary will be effective, then 'inScreen' will be true.

The reason I check 'inScreen' at beginning of 'move method' is that the most time of game is inScreen time. Therefore, in the most of time, my game will not go through off-screen boundary checking in the whole boundary checking processes. Once the star enter into screen, 'move method' only process the code until 'return', which improve the operation efficiency.

```
public void move() {
    if (inScreen) {
        if (x < 2 || x > pfWidth() - 10) {
            xdir = -xdir;
        }
        if (y < 2 || y > pfHeight() - 10)
            ydir = -ydir;
        return;
    }

    if (!inScreen) {
        if (x < -20 || x > pfWidth() + 20) {
            xdir = -xdir;
        }
        if (y < -20 || y > pfHeight() + 20)
            ydir = -ydir;
    }

    if (x > 2 && y > 2 && x < (pfWidth() - 10) && y < (pfHeight() - 10)) {
        inScreen = true;
    }
}
```