Design documentation and design evolution

Superclass
Subclass

Italic – Abstract class Underscore – category Small font – Instances

Red – reusable classes

Current design:

<table>
<tr><td colspan="4">Wingman</td></tr>
<tr><td colspan="3">Game components</td><td>helpers</td></tr>
<tr><td colspan="3">BasicObject - Island</td><td rowspan="4">ResourceTable
Property
Explosion
MunitionUsage
MunitionUsageProperty
CollisionDetector
GameEvent</td></tr>
<tr><td colspan="3">CollidableObject</td></tr>
<tr><td>Munitions
EnemyPlaneDown
EnemyPlaneUp
EnemyPlaneLeft
EnemyPlaneRight
EnemyBullet
MyBulletUp
MyBulletLeft
MyBulletRight</td><td colspan="2">Weapon</td></tr>
<tr><td></td><td>DumpWeapon
EnemyShoot</td><td>SmartWeapon
MyPlane</td></tr>
</table>

Original design:

<table>
<tr><td colspan="6">BasicObject</td></tr>
<tr><td>Island</td><td>Wall</td><td colspan="2">Munitions</td><td colspan="2">Weapon</td></tr>
<tr><td></td><td>Temporary Wall</td><td>EnemyBullet
EnemyAim</td><td>MyMunitions</td><td>EnemyShoot
EnemyTarget
EndBoss</td><td>ControllableWeapon</td></tr>
<tr><td></td><td></td><td></td><td>MyBullet
Shell
Rocket
BouncingBomb</td><td></td><td>MyPlane
Tank</td></tr>
</table>

4 major design changes:

1. Add Resource table – map game components to its predefined information because it is a game project.
2. Focus more on the behavior instead of view to reduce number of classes
   e.g. Munitions is a class.
   Its instances: EnemyPlaneDown, EnemyPlaneUp, EnemyPlaneLeft, EnemyPlaneRight
   EnemyBullet, MyBulletUp, MyBulletLeft, MyBulletRight
3. Reduce responsibilities of BasicObject class and extending it to create a new class - CollidableObject to reduce computation waste in checking collision
4. Mechanism to create bullet

1. Resource table:

Before implementation, I was not aware of the amount of predefined information needed. The predefined information is not just image, explosion, but also the rules of how the game works, like who are the enemies, how much are the damage and reward.

Because of the bulk of information and its varied type, I created several helper classes to group information together, so that in my ResourceTable, I will not have multiple HashMap for each type of information.

Explosion – group animation images and sound file together

MunitionUsageProperty & MunitionUsage – group information of munitions, e.g. type, storage, limited and so on. (In my tank project, I plan to consolidate these two classes.)

Property – group all the predefined information.

Using overloaded constructor to match different classes.

ResourceTable – map objects name to its property

2. Behavior V.S. Data

Focusing more on behavior instead of data is another important reason for my design evolution. In Wingman, many components are the same in behavior, but different in view. Since we are

not doing complicated computer graphics design, it is not necessary to separate the view from model. With the help of naming objects, I greatly reduce the number of classes in my project.

For example, EnemyPlaneDown and EnemyPlaneUp are identical in behavior, but different in data, which are direction and image. (Direction decides image.) It would be a waste to make different classes for them. Here names help differentiate these two and enable mapping the correct properties.

Furthermore, I analyzed game components to summarize common behaviors to create general classes.

BaiscObject – position, direction, draw(), updata()

CollidableObject – collide(), intersects(), explode(), enemyList

Munitions – explode when collision & if bullet, needs to know its owner

Weapon – hit(), fire(), MunitionUsageList

DumpWeapon – fire bullet at a fixed rate

SmartWeapon – react to players

For example, aside from the view, the difference between EnemyPlane and EnemyBullet is their initial position. EnemyPlane's position is decided by the programmer, while EnemyBullet's position is decided by its owner – EnemyShoot. Other than that, they are the same in behavior. (Note that EnemyPlane will collide with MyBullet, while EnemyBullet will not. However, whether collision happen does not mean they do not intersect. Collision happen with two conditions: intersection and one is in another's enemyList, which is predefined. Here is where predefined information comes into play. It is a little annoying. However, with the help of Property class and ResourcesTable, I am able to simplify the problem.) So I use the same class, Munitions, to create EnemyPlane and EnemyBullet. Overloaded constructors are used in Munitions to help set the initial value of EnemyPlane and EnemyBullet. MyBullet and EnemyBullet are similar too.

3. BasicObject → CollidableObject

In my original design, BasicObject is the most basic class in my design and it handles collision (abstract method collide(BasicObject another)). The reason is that in the real world, any object can collide with another. However, since we are designing a game, things do not follow the reality exactly. Like Island will not have collision feature. So there is no need to include collide()

in BasicObject. Based on this, I created CollidableObject which is the subclass of BasicObject. It handles collision, explosion and intersection. Only checking collision between CollidableObject, and also the predefined information - enemyList, it saves a lot of computation resources.

4. Ways to create bullets

In my original design documentation, I didn't think in detail of how to create bullet (where to create it). I was focusing too much on separating responsibilities of classes, which resulting in me using observer pattern to notify Wingman to create bullet instead of passing the information to MyPlane and creating bullet there. Now I realize it is more reasonable for MyPlane to create a bullet, instead of Wingman. I will make this changes later. (This is not done in my submission)