Final Design Document

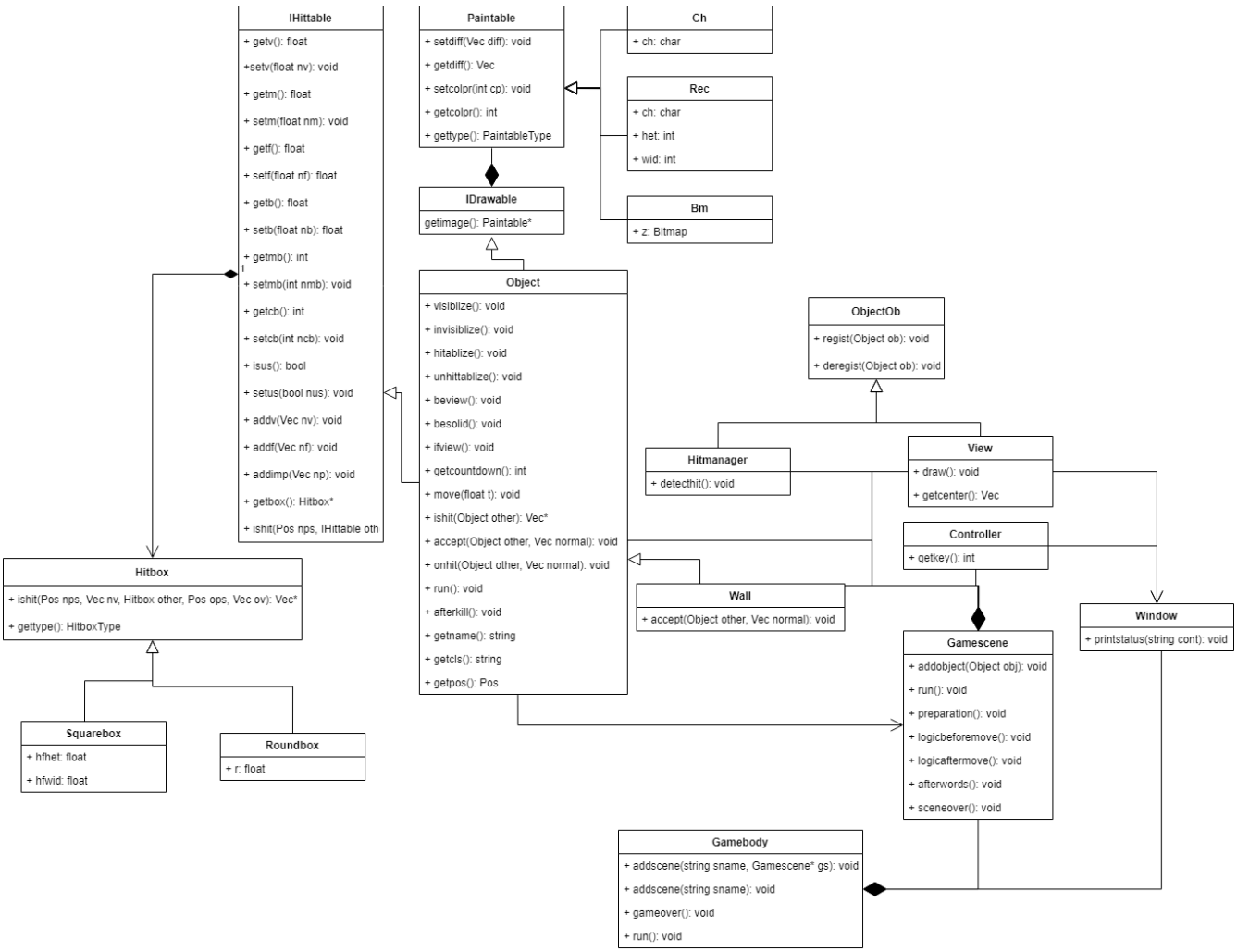
Overview

The project is contains three parts:

- Engine
- Game 1: arlg
- Game 2: Snooker

To make up the project, run "make" command at the main folder, and the two games will be generated in two subfolders. Do not move the executables, or the image in the game may be missing.

Updated UML



Compared with the UML from the beginning, all the public fields of class IHittable has been hidden, and the user can now only interact with them using interfaces. More fields were introduced into IHittable, and so does the class Object. The controller is introduced in the new design.

Design

To make the ncurses library suite the OOP style, the adapter pattern is applied, and the library is mainly wrapped in the Window class.

The collision events is mainly managed with visitor pattern. Different objects can be distinguished using their names and classes, and different collision events can be embedded by overriding the accept method and overloading the onhit method.

The observer pattern is used in collision detection part to inform each objects of the collision event.

Little User Guide

To use the engine to build a game, the user will mainly interact with the three classes: Object, Gamescene and Gamebody.

The engine is based on inheritance.

The user can add various objects with different functions by inheriting from the class Object, and override the default methods includes:

```
virtual void move(timetype t);
virtual void run();
virtual void afterkill();
```

Overriding the move method can change the movement of the object, for example, move along a cycle instead of simple movement constrained by linear physics.

The run() method will be called once per frame, it can be used to do anything. It is default doing nothing.

The afterkill() method will be called after the object is out of its life time. It is default doing nothing.

The user can add different scenes by inheriting from the class Gamescene, and override the default methods includes:

```
virtual void preparation();
virtual void logicbeforemove();
virtual void logicaftermove();
virtual void afterwords();
```

The preparation() method will be called when the scene is going to start.

The logicbeforemove() method will be called before the objects moving.

The logicaftermove() method will be called after the objects moving and the collision detection.

The afterwords() method will be called when the scene is ended.

The pseudo-code for the game cycle will be like:

```
preparation();
while scene goes on {
    logicbeforemove();
    for each object {
```

```
            object.run();
            if the object is marked to be removed {
                object.afterkill();
                remove(object);
            }
        }
        for each object {
            object.move(t);
        }
        detectcollision();
        logicaftermove();
        draw();
    }
    afterwords();
```

There is no reason for the user to override the Gamebody class, and it should be used directly. The user should instantiate exactly one Gamebody class, and interact with it.

The user can use the addobject(name, gamescene) method to register the scene into the repository of the game body, then call addobject(name) method to append the scene to the player list. After calling run() method, the game body will calling each scene in the player list one by one, and terminate if the player list is empty or gameover() method is being called.

Extra Credit Features

1. The fps of the game can be changed dynamically, the user can make use of this feature to create a setting page for the players to change the fps within the game. The feature is added by creating a uniform structure to manage all the values about the fps.
2. Other than the square hitbox, the engine also provides the round hitbox, which can be used to simulate the collision events of round objects. For example, cue balls. This feature is added by creating a new subclass of the class Hitbox, and implementing the collision detection methods between Round hitbox and Square hitbox, and which between Round hitboxes. The visitor pattern is used to distinguish the types of colliding hitboxes.
3. The camera can be moved. The user can call modifycenter() method to shift the camera around, which can be used to create games with relatively big maps, for example, Mario. The second game(Snooker) uses this feature to represents a big pool table(which is 68 * 136), and give the player methods to move around the table. This feature is added by modifying the draw() method.
4. The elasticity of a collision can be changed. One can change the elasticity of two classes of objects' collision by setting the bouncing coefficient of each classes. set bouncing to 1 will incur perfect elastic collisions, while 0 will incur perfect inelastic collisions. Also one can set the mass of each objects, which will influence the effect of the collision. This feature is added by modifying the details of collision.
5. The project is supposed to never leak.

Final question

If I had the chance to start over, I will first spend some time to see how other game engine is built. I have wasted a lot of time reconstructing my collision parts for at least 3 times, and finally after reading the instructions of unity, I successfully constructed a complete collision system, however there's already poor time for me to constructing more extra features.