```cpp
#include "ObservableCollisionDetection.h"

ObservableCollisionDetection* ObservableCollisionDetection::cd = NULL;

ObservableCollisionDetection::ObservableCollisionDetection() {};

void ObservableCollisionDetection::checkCollisions() {
    if (obs_list.empty() != true) {
        //Checking Collisions Algorithm
        //First loop - Grab list x
        for (int i1 = 0; i1 < obs_list.size()-1; i1++) {
            std::vector<SDL_Rect> list_x = obs_list.at(i1)->getColliders();
            //Second Loop - to go through all the rectangles in the list x
            for (int i2 = 0; i2 < list_x.size(); i2++) {
                //Only the Player class has 2 colliders, therefore i2 won't count higher than 2
                //Third loop - Grab next list y
                for (int i3 = i1+1; i3 < obs_list.size(); i3++) {
                    std::vector<SDL_Rect> list_y = obs_list.at(i3)->getColliders();
                    //Fourth Loop - to go through all the rectangles in list y
                    for (int i4 = 0; i4 < list_y.size(); i4++) {
                        //Only the Player class has 2 colliders, therefore i4 won't count higher than 2
                        //compare first_list[i2] with second_list[i4]
                        if (collisionBetween(&list_x.at(i2), &list_y.at(i4)) == true) {
                            // If i4 == 1 it means that list_y.at(i4) is a sword collider
                            // If i2 == 1 it means that list_x.at(i2) is a sword collider
                            //enum collider_types { BODY, SWORD, BOTTLE, BOTTLEHARMLESS };
                            //update(collided_with, own_collider, rectangle with which the collision appeared);
                            obs_list.at(i1)->update(obs_list.at(i3)->getColliderType(i4), obs_list.at(i1)->getColliderType(i2), list_y.at(i4));
                            obs_list.at(i3)->update(obs_list.at(i1)->getColliderType(i2), obs_list.at(i3)->getColliderType(i4), list_x.at(i2));
                        }
                    }
                }
            }
        }
    }
};

bool ObservableCollisionDetection::intersect(int start1, int end1, int start2, int end2) {
    bool intersection = true;
```

```
40      if (end1 < start2) {
41          intersection = false;
42      }
43      else if (end2 < start1) {
44          intersection = false;
45      }
46      return intersection;
47  };
48
49  bool ObservableCollisionDetection::collisionBetween(SDL_Rect* a, SDL_Rect*
      b) {
50      bool success = false;
51
52      if ((a != NULL) && (b != NULL)) {
53          //Check if collided on x or y axis
54          bool x_axis = false;
55          bool y_axis = false;
56
57          //Check x_axis
58          if (intersect(a->x, a->x + a->w, b->x, b->x + b->w) == true) {
59              x_axis = true;
60              //Check y_axis
61              if (intersect(a->y, a->y + a->h, b->y, b->y + b->h) == true) {
62                  y_axis = true;
63              }
64          }
65
66          //Check for actual collision
67          if ((x_axis == true) && (y_axis == true)) {
68              success = true;
69          }
70      }
71
72      return success;
73  };
74
75  ObservableCollisionDetection* ObservableCollisionDetection::getInstance() {
76      if (cd == NULL) {
77          cd = new ObservableCollisionDetection();
78      }
79
80      return cd;
81  };
82
83  ObservableCollisionDetection::~ObservableCollisionDetection() {
84      cd = NULL;
85  };
86
```