Spike: Task 22.P
Title: Collisions

Author: Mitchell Wright, 100595153

Goals / deliverables:

To demonstrate .

Items created during task:

- Code, see: \22 - Spike – Collisions\SDL2

Technologies, Tools, and Resources used:

- Visual Studio 2022
- SourceTree
- GitHub
- SDL2
- Lecture 6.2 – 2D Collision Tests

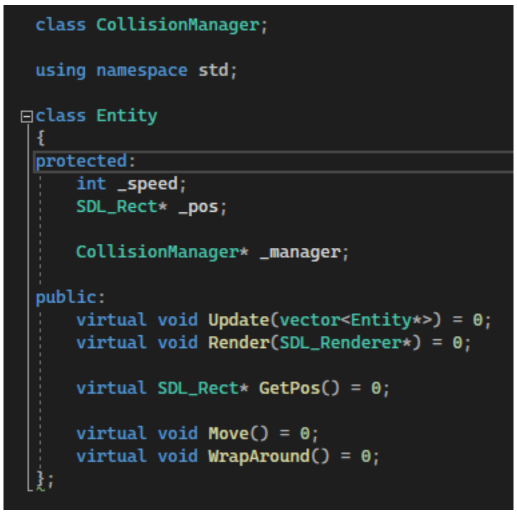
Tasks undertaken:

- Implement Moving Entities
- Implement Collision Data Structure
- Implement the Collision Tests
- Commit to Git

What we found out:

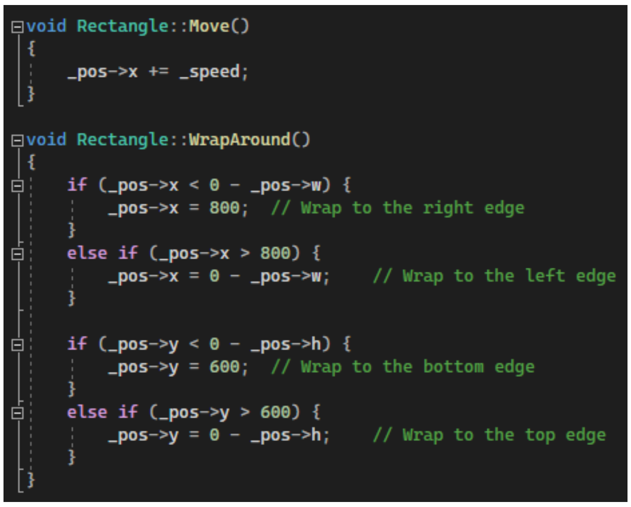
1. Implement Moving Entities:

In order for everything to work smoothly later on, I created an Entity super-class with most fo the main functionality needed, then created a Circle and Rectangle class.



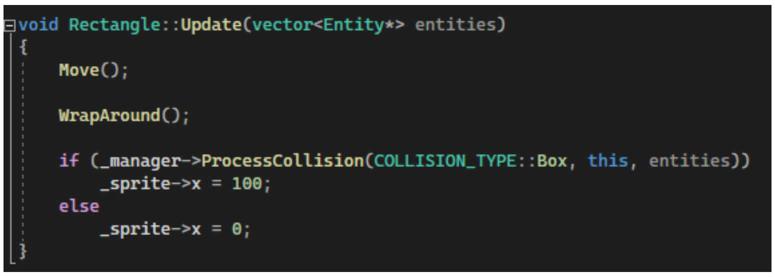
```cpp
class CollisionManager;

using namespace std;

class Entity
{
protected:
    int _speed;
    SDL_Rect* _pos;

    CollisionManager* _manager;

public:
    virtual void Update(vector<Entity*>) = 0;
    virtual void Render(SDL_Renderer*) = 0;

    virtual SDL_Rect* GetPos() = 0;

    virtual void Move() = 0;
    virtual void WrapAround() = 0;
};
```

Move and Wrap Around are pretty simple, basically if the entity goes beyond the bounds of the screen, move it back to the other side, otherwise it moves from left to right at whatever speed is set on instantiation.



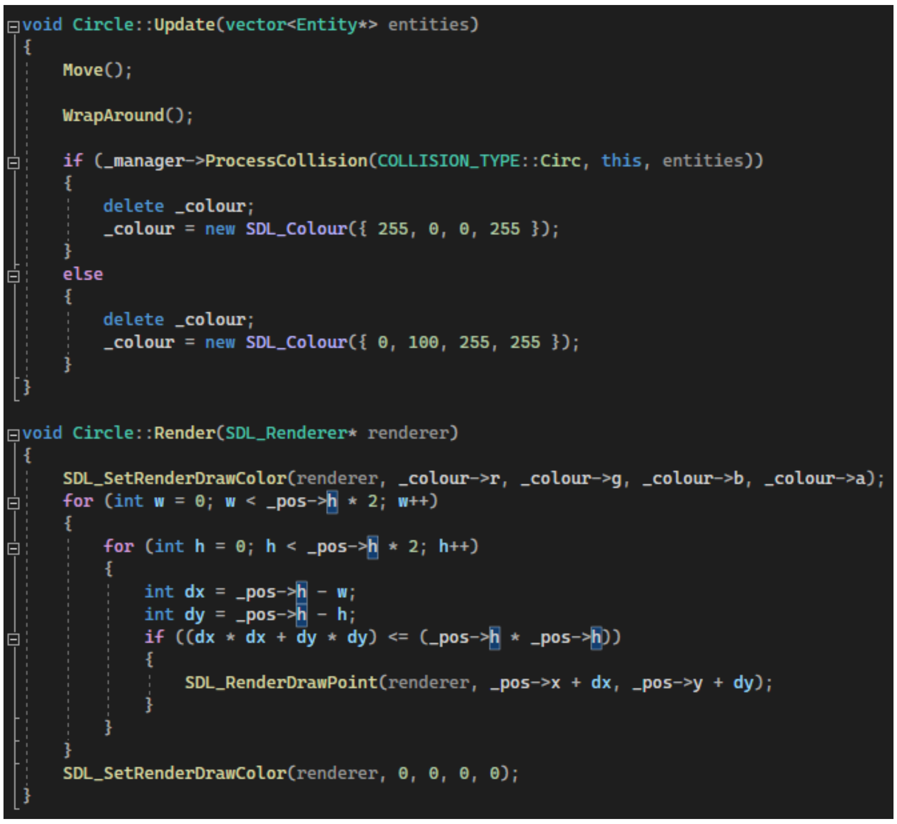
```cpp
void Rectangle::Move()
{
    _pos->x += _speed;
}

void Rectangle::WrapAround()
{
    if (_pos->x < 0 - _pos->w) {
        _pos->x = 800;  // Wrap to the right edge
    }
    else if (_pos->x > 800) {
        _pos->x = 0 - _pos->w;    // Wrap to the left edge
    }

    if (_pos->y < 0 - _pos->h) {
        _pos->y = 600;  // Wrap to the bottom edge
    }
    else if (_pos->y > 600) {
        _pos->y = 0 - _pos->h;    // Wrap to the top edge
    }
}
```

Update and render for the rectangles work basically the same as in the previous Sprite task.



```cpp
void Rectangle::Update(vector<Entity*> entities)
{
    Move();

    WrapAround();

    if (_manager->ProcessCollision(COLLISION_TYPE::Box, this, entities))
        _sprite->x = 100;
    else
        _sprite->x = 0;
}
```

Circles work a bit differently, being drawn rather than a rendered sprite. This also means that update changes colours rather than changing sprite positioning.

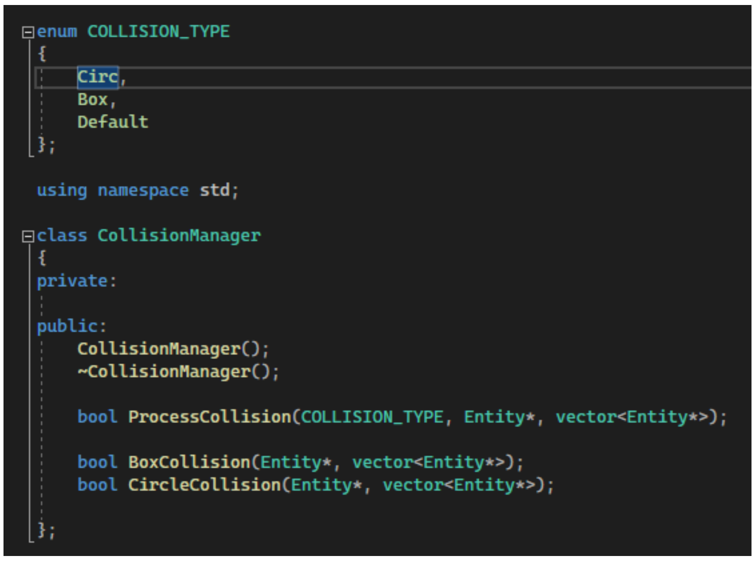


```cpp
void Circle::Update(vector<Entity*> entities)
{
    Move();

    WrapAround();

    if (_manager->ProcessCollision(COLLISION_TYPE::Circ, this, entities))
    {
        delete _colour;
        _colour = new SDL_Colour({ 255, 0, 0, 255 });
    }
    else
    {
        delete _colour;
        _colour = new SDL_Colour({ 0, 100, 255, 255 });
    }
}

void Circle::Render(SDL_Renderer* renderer)
{
    SDL_SetRenderDrawColor(renderer, _colour->r, _colour->g, _colour->b, _colour->a);
    for (int w = 0; w < _pos->h * 2; w++)
    {
        for (int h = 0; h < _pos->h * 2; h++)
        {
            int dx = _pos->h - w;
            int dy = _pos->h - h;
            if ((dx * dx + dy * dy) <= (_pos->h * _pos->h))
            {
                SDL_RenderDrawPoint(renderer, _pos->x + dx, _pos->y + dy);
            }
        }
    }
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0);
}
```

Objects are then run in an update/render loop in main as part of a vector.

2. Implementing the Collision Manager:

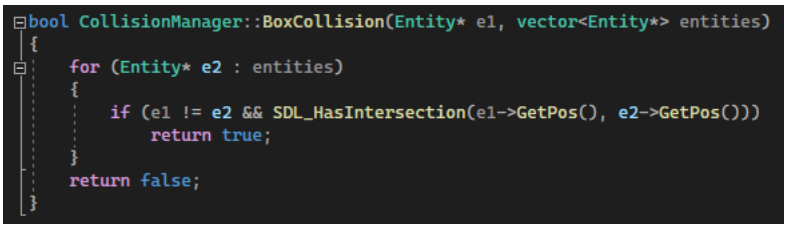
The task sheet for this spike said something about making it extensible. Didn't really understand what that meant, so I made a collision manager with different functions for circle or box collision and an enum to select which one was appropriate at any given time.



```cpp
enum COLLISION_TYPE
{
    Circ,
    Box,
    Default
};

using namespace std;

class CollisionManager
{
private:

public:
    CollisionManager();
    ~CollisionManager();

    bool ProcessCollision(COLLISION_TYPE, Entity*, vector<Entity*>);

    bool BoxCollision(Entity*, vector<Entity*>);
    bool CircleCollision(Entity*, vector<Entity*>);

};
```

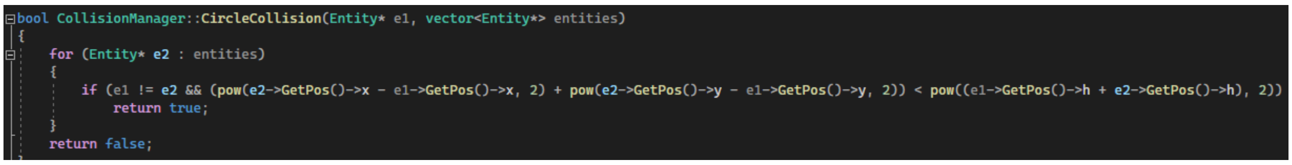
3. Implementing Collision Tests:

The actual collision tests themselves weren't too tricky. For Rectangles, SDL2 has a built-in function allowing for finding an intersection.



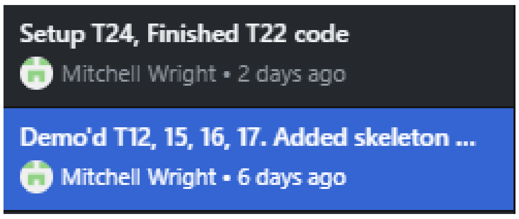
```cpp
bool CollisionManager::BoxCollision(Entity* e1, vector<Entity*> entities)
{
    for (Entity* e2 : entities)
    {
        if (e1 != e2 && SDL_HasIntersection(e1->GetPos(), e2->GetPos()))
            return true;
    }
    return false;
}
```

Circles are a bit more fiddly, and I essentially converted the pseudo code provided in the lecture into C++ and used that to check if the distance between circles was less than their combined radii.



```cpp
bool CollisionManager::CircleCollision(Entity* e1, vector<Entity*> entities)
{
    for (Entity* e2 : entities)
    {
        if (e1 != e2 && (pow(e2->GetPos()->x - e1->GetPos()->x, 2) + pow(e2->GetPos()->y - e1->GetPos()->y, 2)) < pow((e1->GetPos()->h + e2->GetPos()->h), 2))
            return true;
    }
    return false;
```

4. Commit to Git:



Setup T24, Finished T22 code
Mitchell Wright • 2 days ago

Demo'd T12, 15, 16, 17. Added skeleton ...
Mitchell Wright • 6 days ago