Task 22 - Spike Summary Report

💡 **Spike:** Task_22
Title: Collisions
Author: Thomas Horsley, 103071494

Goals & Deliverables

Aim: Implement both circular and axis aligned rectangular collision detection mechanisms using SDL2

Deliverables:

- Functional code circular and axis aligned box collision detection
- Git commit history
- Spike Report

Technology, Tools and Resources

Tech and Tools

💡 The project was scripted in C++ 17 using Visual Studio Community 2022.

UML's and charts are made with *www.Lucidchart.com*

Source control is handled using Git.

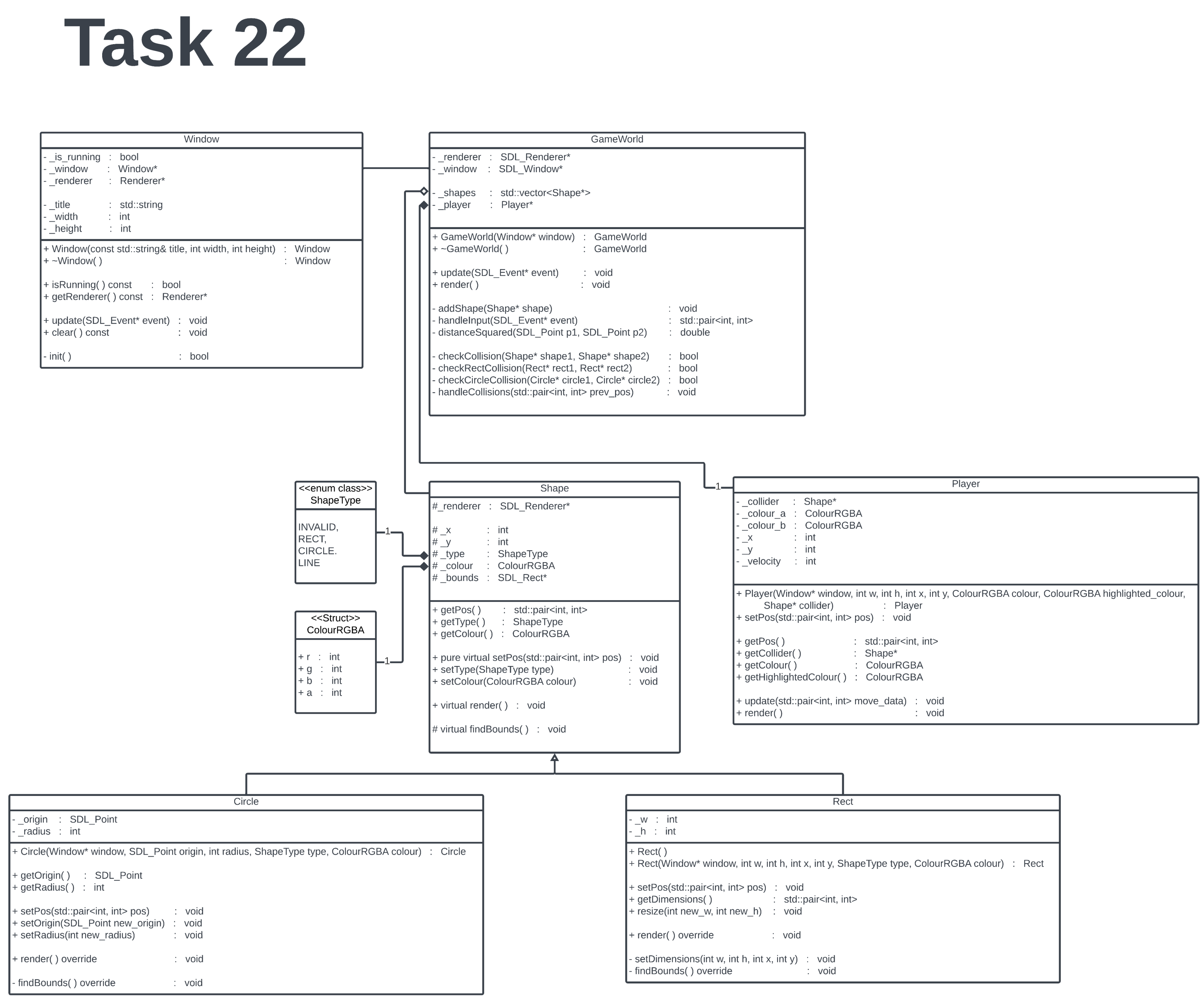
Resources

- Lazyfoo Collision Detection: https://lazyfoo.net/tutorials/SDL/27_collision_detection/index.php
- Building a 2D Primatives Library: See: https://www.youtube.com/watch?v=EnKZnwbgn-U&t=1816s

Task 22 - Spike Summary Report

Planning

Diagrams and Charts


Task 22
Window
- _is_running : bool
- _window : Window*
- _renderer : Renderer*
- _title : std::string
- _width : int
- _height : int
+ Window(const std::string& title, int width, int height) : Window
+ ~Window( ) : Window
+ isRunning( ) const : bool
+ getRenderer( ) const : Renderer*
+ update(SDL_Event* event) : void
+ clear( ) const : void
- init( ) : bool
GameWorld
- _renderer : SDL_Renderer*
- _window : SDL_Window*
- _shapes : std::vector<Shape*>
- _player : Player*
+ GameWorld(Window* window) : GameWorld
+ ~GameWorld( ) : GameWorld
+ update(SDL_Event* event) : void
+ render( ) : void
- addShape(Shape* shape) : void
- handleInput(SDL_Event* event) : std::pair<int, int>
- distanceSquared(SDL_Point p1, SDL_Point p2) : double
- checkCollision(Shape* shape1, Shape* shape2) : bool
- checkRectCollision(Rect* rect1, Rect* rect2) : bool
- checkCircleCollision(Circle* circle1, Circle* circle2) : bool
- handleCollisions(std::pair<int, int> prev_pos) : void
<<enum class>> ShapeType
INVALID, RECT, CIRCLE. LINE
1
<<Struct>> ColourRGBA
+ r : int
+ g : int
+ b : int
+ a : int
1
Shape
# _renderer : SDL_Renderer*
# _x : int
# _y : int
# _type : ShapeType
# _colour : ColourRGBA
# _bounds : SDL_Rect*
+ getPos( ) : std::pair<int, int>
+ getType( ) : ShapeType
+ getColour( ) : ColourRGBA
+ pure virtual setPos(std::pair<int, int> pos) : void
+ setType(ShapeType type) : void
+ setColour(ColourRGBA colour) : void
+ virtual render( ) : void
# virtual findBounds( ) : void
1
Player
- _collider : Shape*
- _colour_a : ColourRGBA
- _colour_b : ColourRGBA
- _x : int
- _y : int
- _velocity : int
+ Player(Window* window, int w, int h, int x, int y, ColourRGBA colour, ColourRGBA highlighted_colour, Shape* collider) : Player
+ setPos(std::pair<int, int> pos) : void
+ getPos( ) : std::pair<int, int>
+ getCollider( ) : Shape*
+ getColour( ) : ColourRGBA
+ getHighlightedColour( ) : ColourRGBA
+ update(std::pair<int, int> move_data) : void
+ render( ) : void
Circle
- _origin : SDL_Point
- _radius : int
+ Circle(Window* window, SDL_Point origin, int radius, ShapeType type, ColourRGBA colour) : Circle
+ getOrigin( ) : SDL_Point
+ getRadius( ) : int
+ setPos(std::pair<int, int> pos) : void
+ setOrigin(SDL_Point new_origin) : void
+ setRadius(int new_radius) : void
+ render( ) override : void
- findBounds( ) override : void
Rect
- _w : int
- _h : int
+ Rect( )
+ Rect(Window* window, int w, int h, int x, int y, ShapeType type, ColourRGBA colour) : Rect
+ setPos(std::pair<int, int> pos) : void
+ getDimensions( ) : std::pair<int, int>
+ resize(int new_w, int new_h) : void
+ render( ) override : void
- setDimensions(int w, int h, int x, int y) : void
- findBounds( ) override : void

Class Notes

Class / Method	Notes
Window	Acts as a wrapper for the SDL_Renderer and SDL_Window, has ownership over these objects. Has an update function which checks for window events.
GameWorld	Owns and is responsible for managing all of the Entities contained within the game world. For this task, the GameWorld has the additional responsibility of handling all the collision events.
Player	For this project, the player is a wrapper for a 2D collider with some movement spice appended. It also has the ability to be notified whenever a collision occurs.
Shape	Contains an SDL_Rect to define it’s rendering bounds and holds information on it’s colour and type. Member variables are accessible through properties and each shape has the ability to calculate it’s bounds.

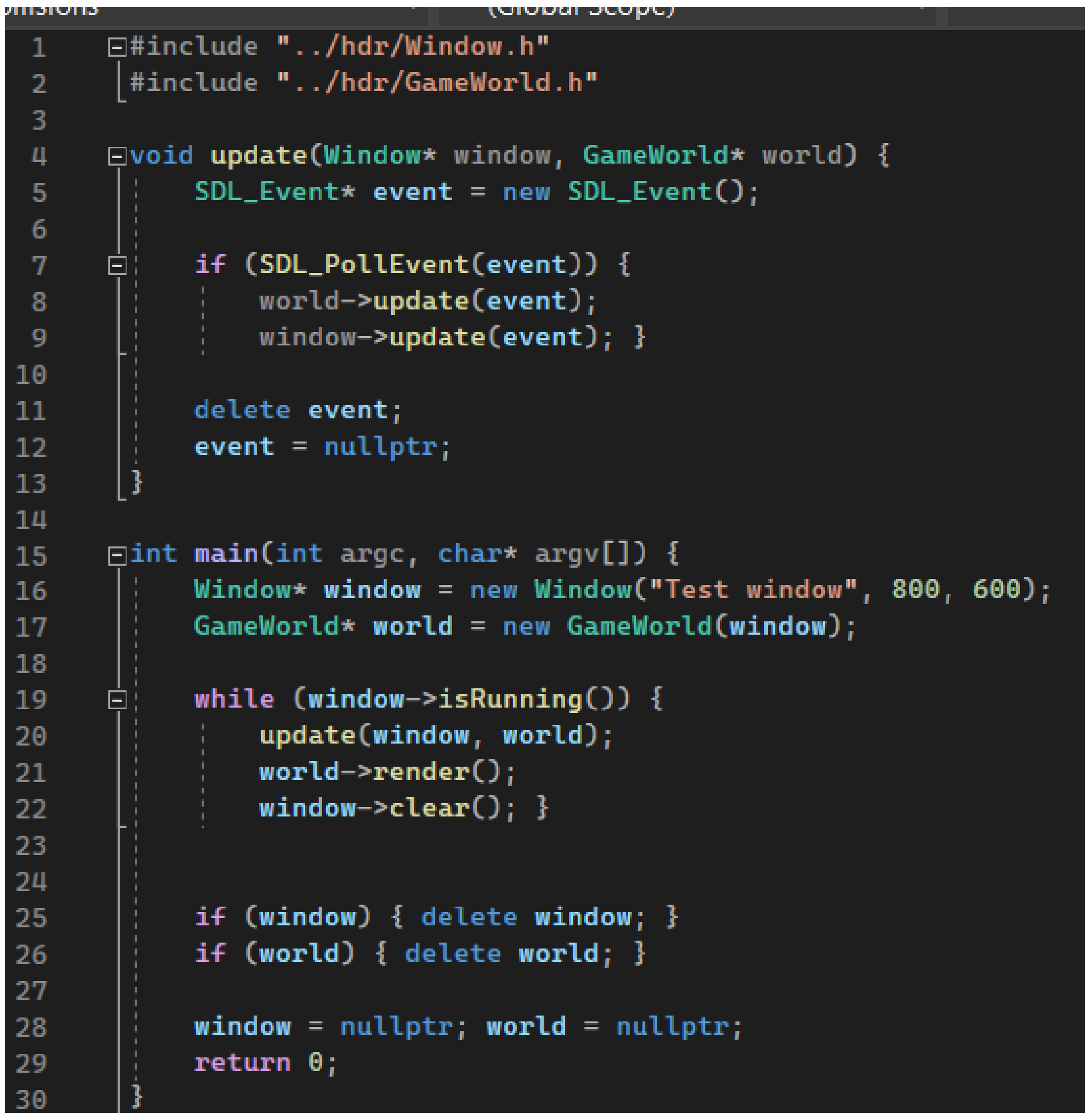
GameWorld::checkCollisions( )	Will take two collision meshes of any type and checks if they’re intersecting, the method of detection is relative to the types of meshes being passed (enum class value). For this example I’ve chosen to use a the most trivial geometric solutions for the collision detection.
GameWorld::handleCollisions( )	If a collision is detected then any updates will be processed within this method. For our cases, the player will be reset back to it’s pre-collision and it’s colour will alternate between two states.
GameWorld::update( )	Handle the users input, update the player and check for collisions against the player and other shapes.
GameWorld::render( )	Render the player first. Make sure the players collider is the 0th element of the _shapes array then call Shape::Render( ) from 1nd element until the end.
Player::update( )	Takes some movement data from the GameWorld::HandleInput( ) and translates that into horizontal or vertical movement. Must recalculate it’s bounds on each move.
Shape::findBounds( )	Calculates the x, y, w, h for the shapes bounding box. This box is used for defining a rendering area.

Implementation

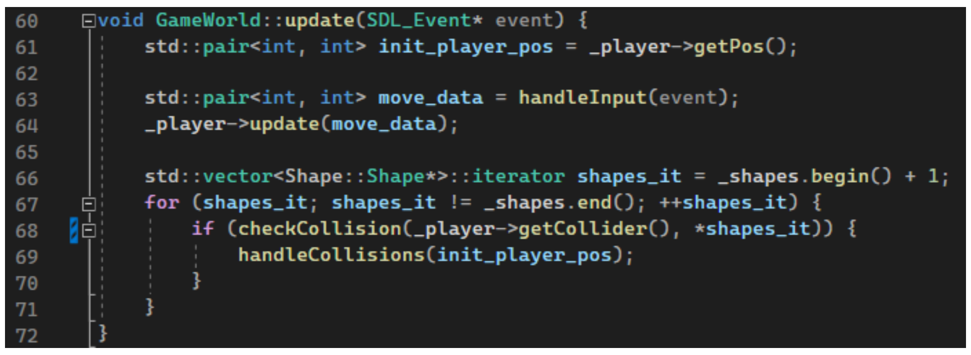
Code

The main event loop runs an update( ) method which lies outside of both the world and the window. This was to resolve any event collisions and errors which may result from simultaneous event polling.

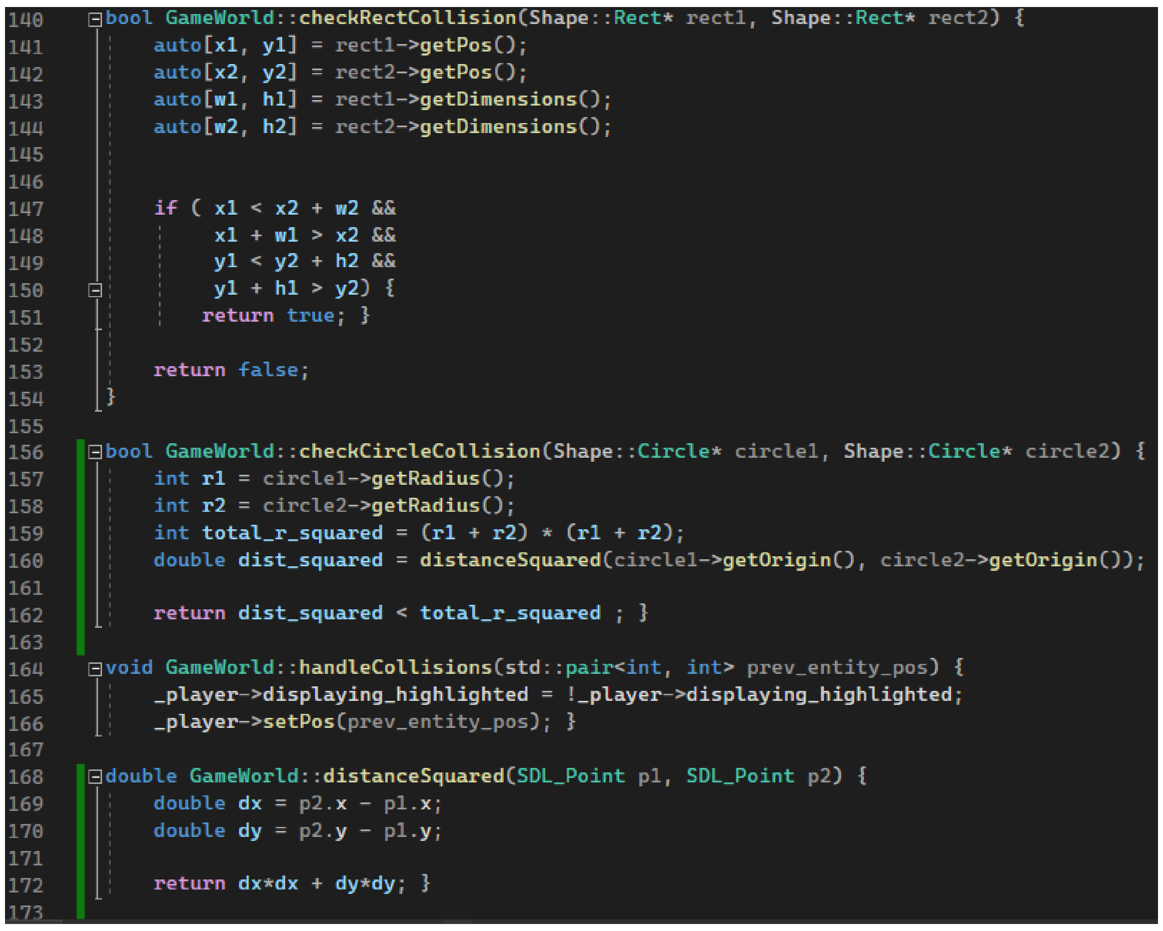
The Window is a wrapper class for the SDL_Window and allows for basic user interaction with the window as well as background rendering.



```cpp
#include "../hdr/Window.h"
#include "../hdr/GameWorld.h"

void update(Window* window, GameWorld* world) {
    SDL_Event* event = new SDL_Event();

    if (SDL_PollEvent(event)) {
        world->update(event);
        window->update(event); }

    delete event;
    event = nullptr;
}

int main(int argc, char* argv[]) {
    Window* window = new Window("Test window", 800, 600);
    GameWorld* world = new GameWorld(window);

    while (window->isRunning()) {
        update(window, world);
        world->render();
        window->clear(); }


    if (window) { delete window; }
    if (world) { delete world; }

    window = nullptr; world = nullptr;
    return 0;
}
```



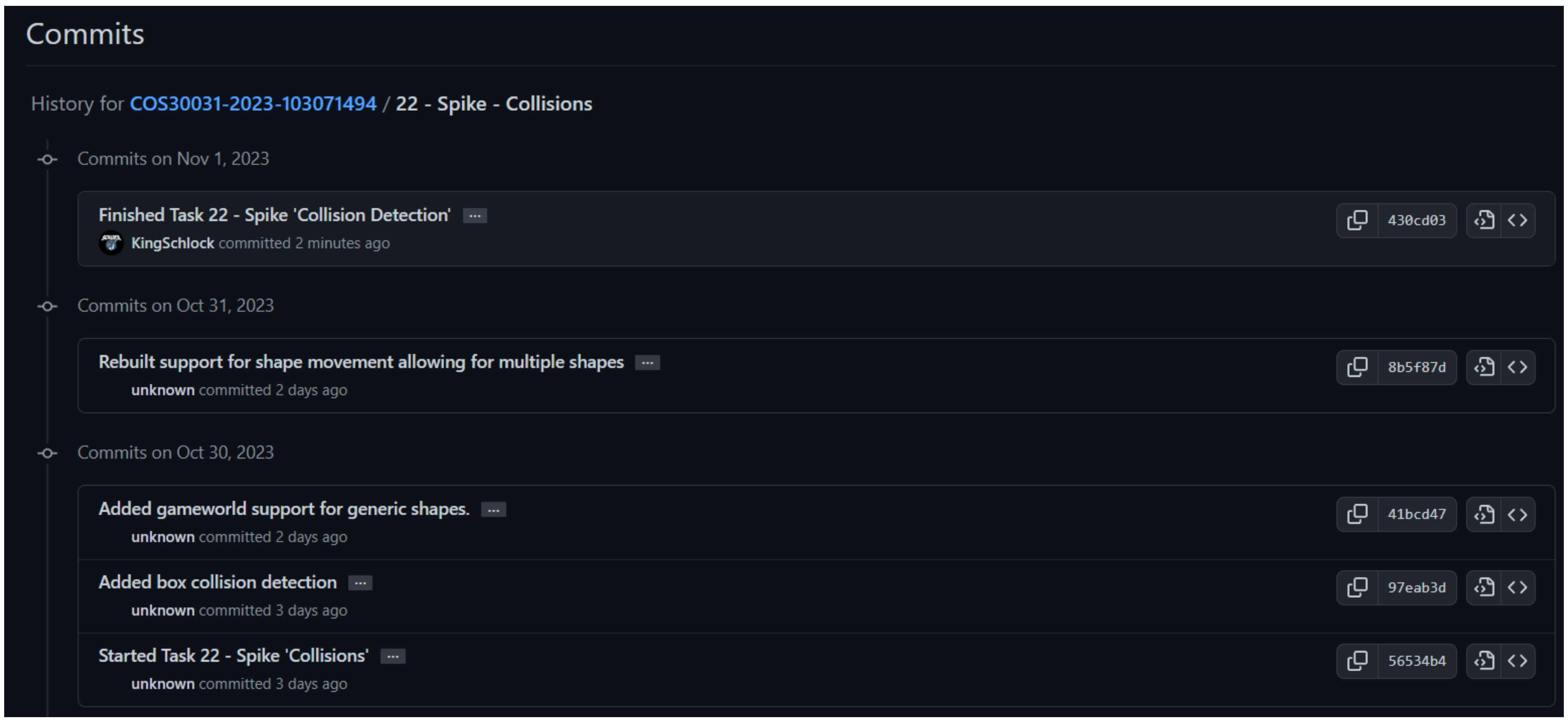
```cpp
void GameWorld::update(SDL_Event* event) {
    std::pair<int, int> init_player_pos = _player->getPos();

    std::pair<int, int> move_data = handleInput(event);
    _player->update(move_data);

    std::vector<Shape::Shape*>::iterator shapes_it = _shapes.begin() + 1;
    for (shapes_it; shapes_it != _shapes.end(); ++shapes_it) {
        if (checkCollision(_player->getCollider(), *shapes_it)) {
            handleCollisions(init_player_pos);
        }
    }
}
```



```cpp
bool GameWorld::checkRectCollision(Shape::Rect* rect1, Shape::Rect* rect2) {
    auto[x1, y1] = rect1->getPos();
    auto[x2, y2] = rect2->getPos();
    auto[w1, h1] = rect1->getDimensions();
    auto[w2, h2] = rect2->getDimensions();


    if ( x1 < x2 + w2 &&
         x1 + w1 > x2 &&
         y1 < y2 + h2 &&
         y1 + h1 > y2) {
        return true; }

    return false;
}

bool GameWorld::checkCircleCollision(Shape::Circle* circle1, Shape::Circle* circle2) {
    int r1 = circle1->getRadius();
    int r2 = circle2->getRadius();
    int total_r_squared = (r1 + r2) * (r1 + r2);
    double dist_squared = distanceSquared(circle1->getOrigin(), circle2->getOrigin());

    return dist_squared < total_r_squared ; }

void GameWorld::handleCollisions(std::pair<int, int> prev_entity_pos) {
    _player->displaying_highlighted = !_player->displaying_highlighted;
    _player->setPos(prev_entity_pos); }

double GameWorld::distanceSquared(SDL_Point p1, SDL_Point p2) {
    double dx = p2.x - p1.x;
    double dy = p2.y - p1.y;

    return dx*dx + dy*dy; }
```

Collision detection mechanisms. For less trivial implementations a CollisionSystem should be created.

Commit History



Commits

History for COS30031-2023-103071494 / 22 - Spike - Collisions

Commits on Nov 1, 2023

Finished Task 22 - Spike 'Collision Detection' ···
KingSchlock committed 2 minutes ago — 430cd03

Commits on Oct 31, 2023

Rebuilt support for shape movement allowing for multiple shapes ···
unknown committed 2 days ago — 8b5f87d

Commits on Oct 30, 2023

Added gameworld support for generic shapes. ···
unknown committed 2 days ago — 41bcd47

Added box collision detection ···
unknown committed 3 days ago — 97eab3d

Started Task 22 - Spike 'Collisions' ···
unknown committed 3 days ago — 56534b4

What was Learned?

💡 This spike taught me how to manage collisions between entities and use those collisions to trigger events and transfer data between entities and their managers.