Tedm

Theodore Ahlfeld
Matthew Haigh
David Watkins

Background

Opposite of foreground

Game Library Market

- Owned by big players
- Many different language solutions
- Have many different skillsets
- Not Portable

- Most of the time: overkill

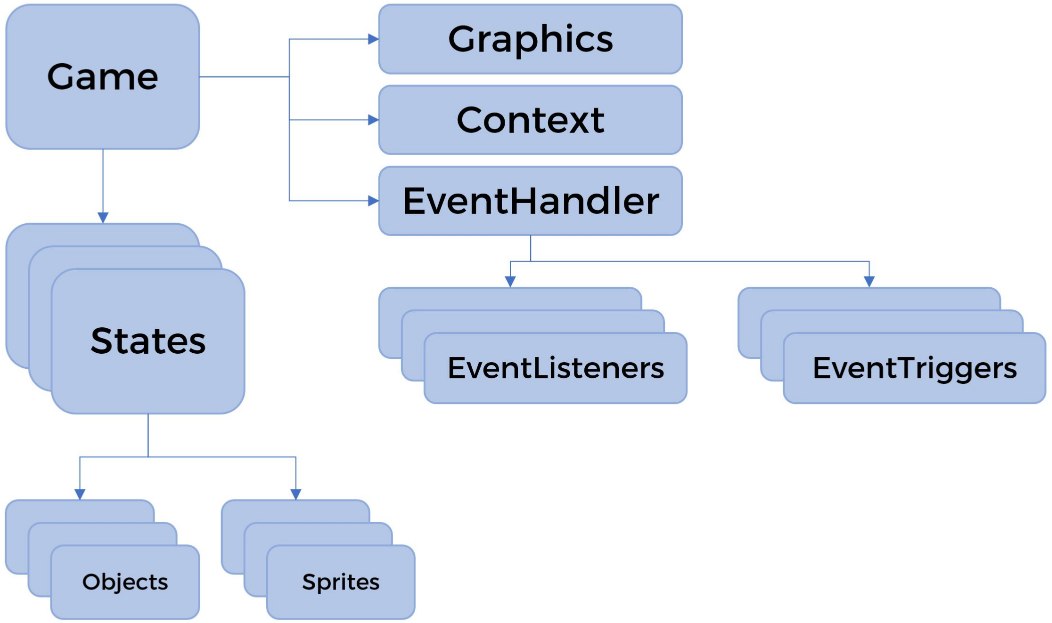
Distilling the Market

- 2D game genre
- Basic event listeners
- Sprites
- States

Design Overview

This is where the fun begins

System Overview



Graphics

- Encapsulate SDL renderer
- All drawable objects use Graphics
- Hide SDL pointers

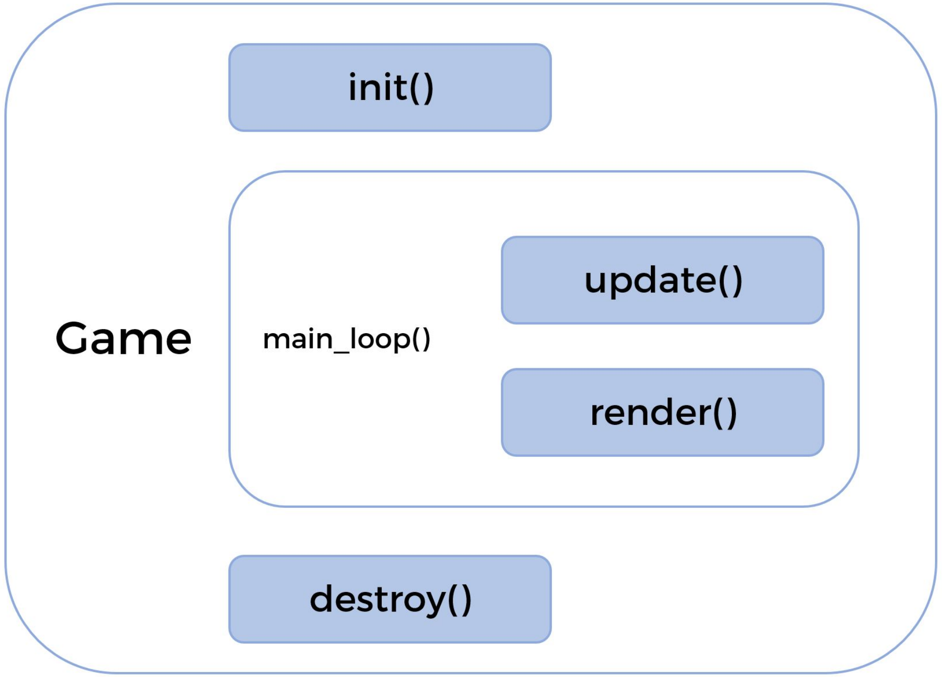
Objects

- Manage position on x,y grid
- Able to draw itself
- All actors extend Object

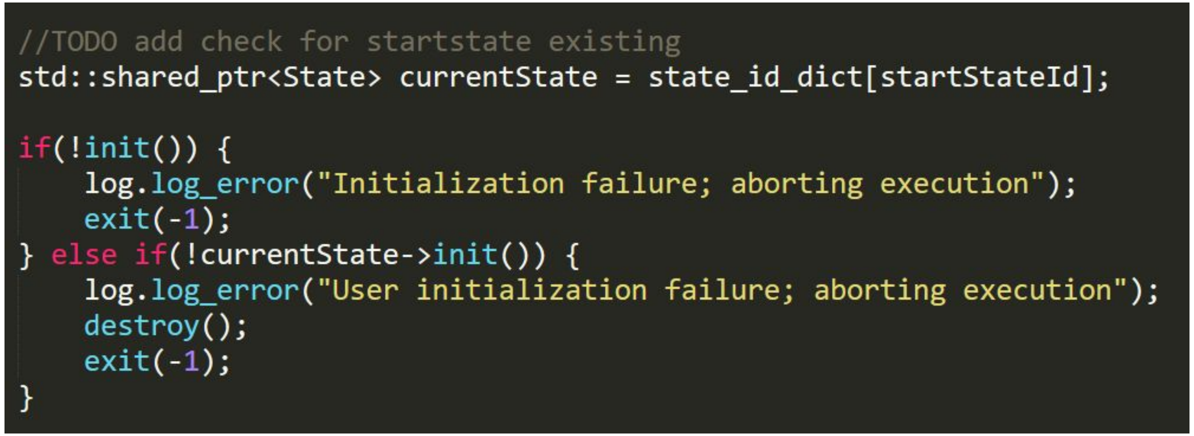
Context

- Persistent object
- Easily extended
- Set window width or fps

Game Object Life Cycle

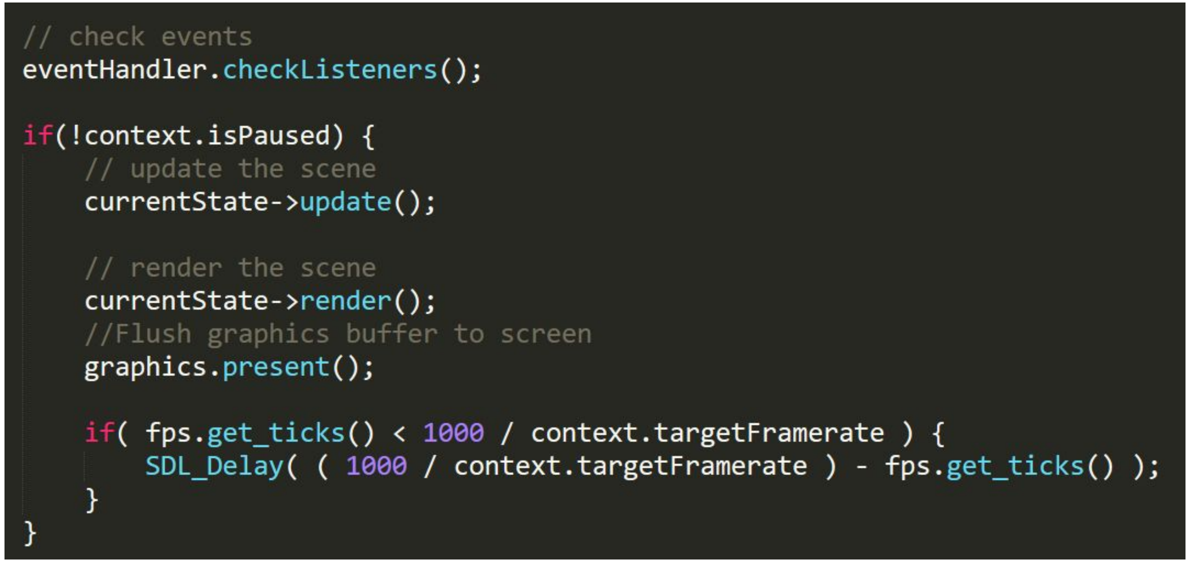


Game Object Init



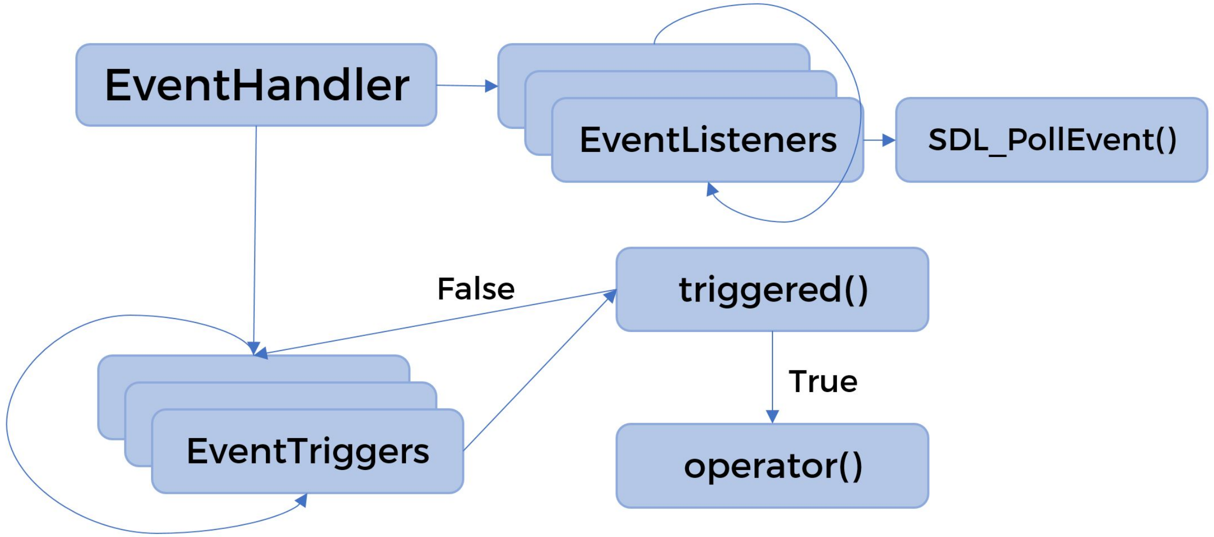
```
//TODO add check for startstate existing
std::shared_ptr<State> currentState = state_id_dict[startStateId];

if(!init()) {
    log.log_error("Initialization failure; aborting execution");
    exit(-1);
} else if(!currentState->init()) {
    log.log_error("User initialization failure; aborting execution");
    destroy();
    exit(-1);
}
```

Game Render/Update

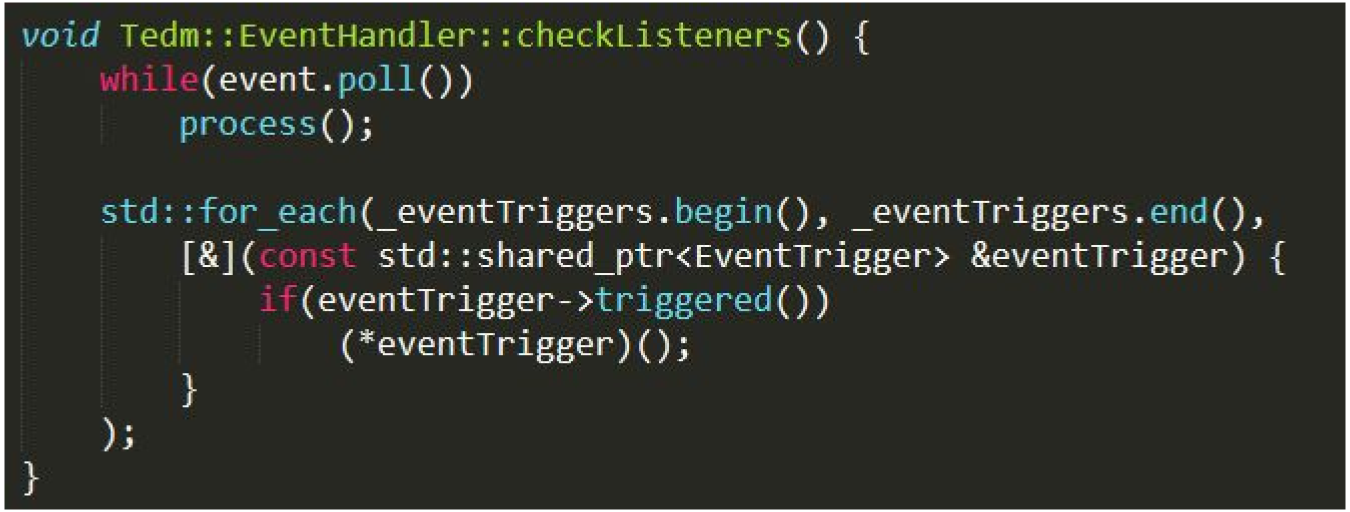


```
// check events
eventHandler.checkListeners();

if(!context.isPaused) {
    // update the scene
    currentState->update();

    // render the scene
    currentState->render();
    //Flush graphics buffer to screen
    graphics.present();

    if( fps.get_ticks() < 1000 / context.targetFramerate ) {
        SDL_Delay( ( 1000 / context.targetFramerate ) - fps.get_ticks() );
    }
}
```

Event Handling

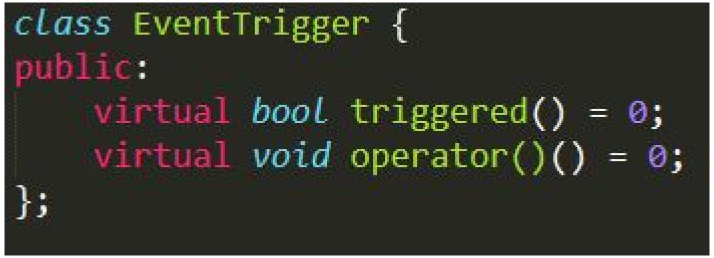


EventHandler Loop



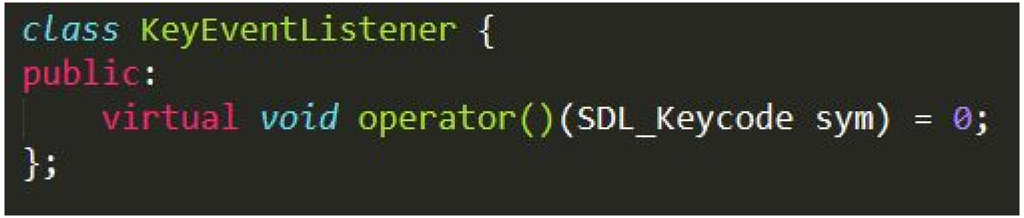
```
void Tedm::EventHandler::checkListeners() {
    while(event.poll())
        process();

    std::for_each(_eventTriggers.begin(), _eventTriggers.end(),
        [&](const std::shared_ptr<EventTrigger> &eventTrigger) {
            if(eventTrigger->triggered())
                (*eventTrigger)();
        }
    );
}
```

EventTrigger



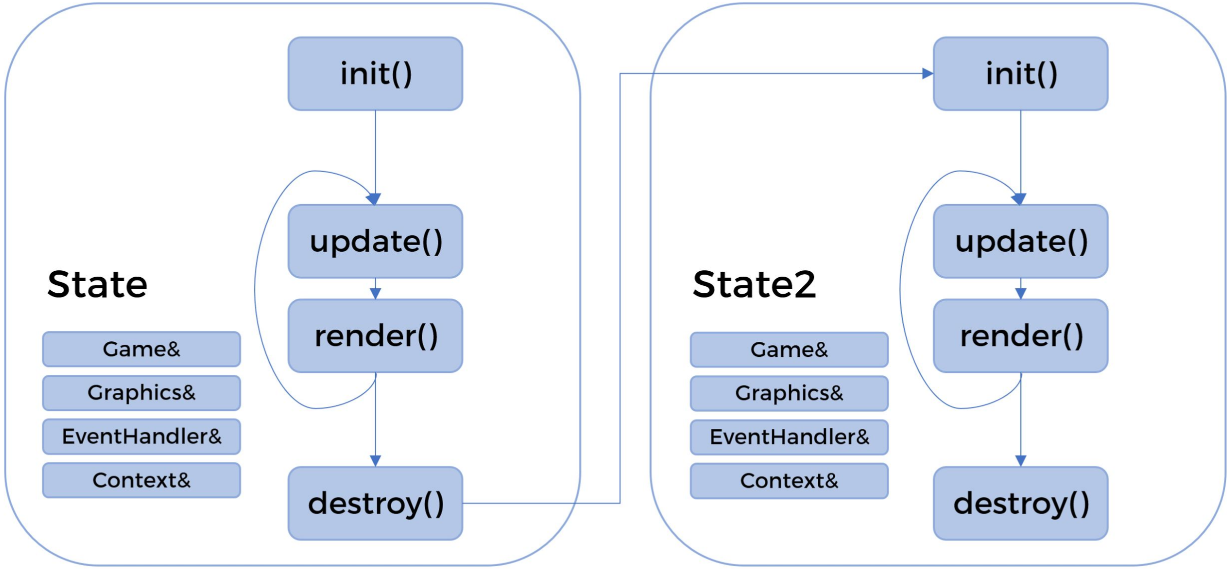
```
class EventTrigger {
public:
    virtual bool triggered() = 0;
    virtual void operator()() = 0;
};
```

EventListener



```
class KeyEventListener {
public:
    virtual void operator()(SDL_Keycode sym) = 0;
};
```

State Life Cycle



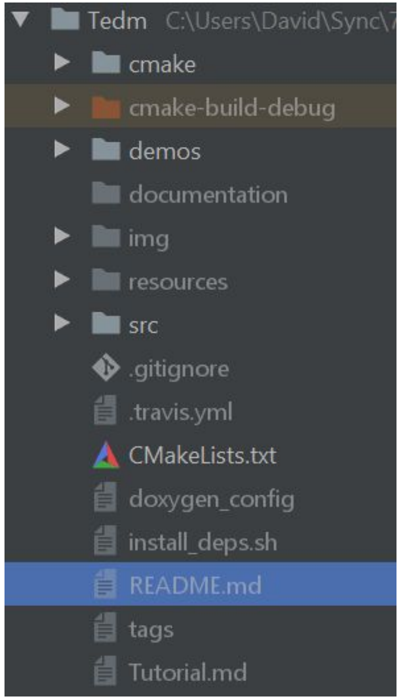
Build Environment

Yakkety Yak

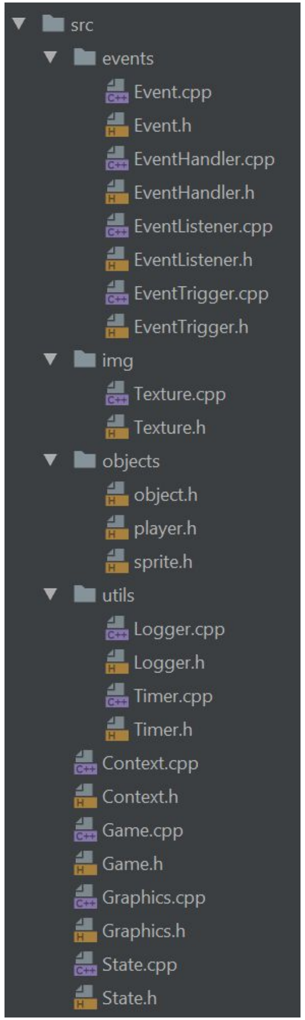
Commonly Used Tools

- Cmake 3.7
- g++ 6.2
- Ubuntu 16.10
- SDL2 and SDL2_image
- Travis-CI
- CLion
- vi

File Structure

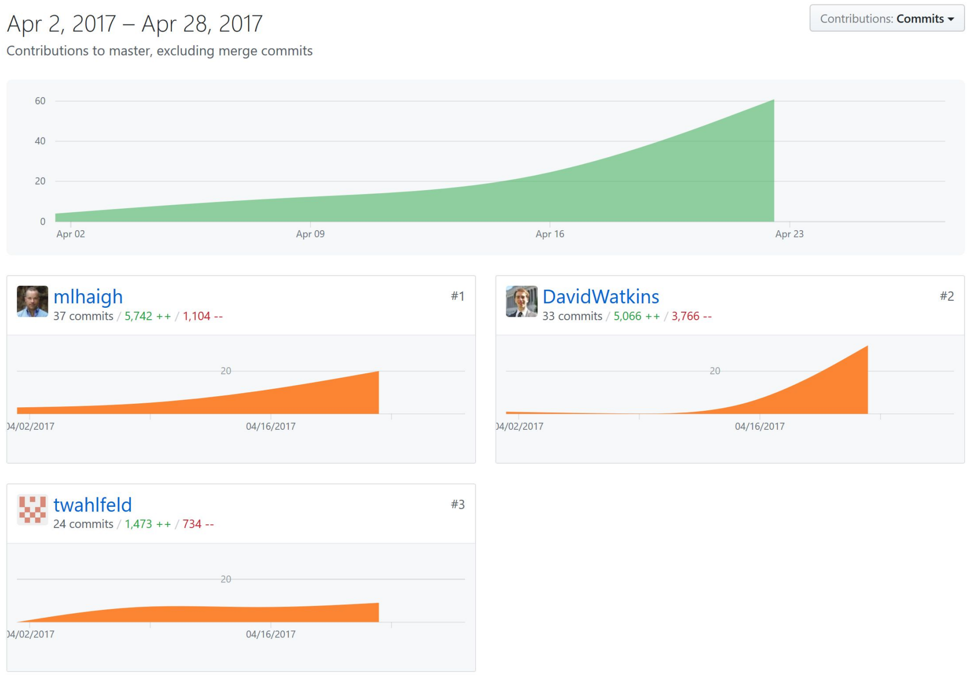


```
▼ Tedm  C:\Users\David\Sync\
   ▶ cmake
   ▶ cmake-build-debug
   ▶ demos
     documentation
   ▶ img
   ▶ resources
   ▶ src
     .gitignore
     .travis.yml
     CMakeLists.txt
     doxygen_config
     install_deps.sh
     README.md
     tags
     Tutorial.md
```



```
▼ src
   ▼ events
        Event.cpp
        Event.h
        EventHandler.cpp
        EventHandler.h
        EventListener.cpp
        EventListener.h
        EventTrigger.cpp
        EventTrigger.h
   ▼ img
        Texture.cpp
        Texture.h
   ▼ objects
        object.h
        player.h
        sprite.h
   ▼ utils
        Logger.cpp
        Logger.h
        Timer.cpp
        Timer.h
     Context.cpp
     Context.h
     Game.cpp
     Game.h
     Graphics.cpp
     Graphics.h
     State.cpp
     State.h
```

Git Commit History



Apr 2, 2017 – Apr 28, 2017

Contributions to master, excluding merge commits

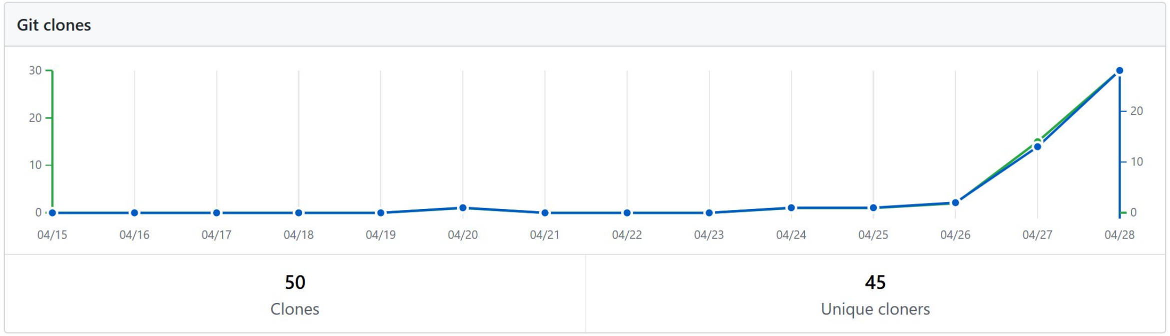
Contributions: **Commits**

mlhaigh #1
37 commits / 5,742 ++ / 1,104 --

DavidWatkins #2
33 commits / 5,066 ++ / 3,766 --

twahlfeld #3
24 commits / 1,473 ++ / 734 --

Clone Fiasco



50
Clones

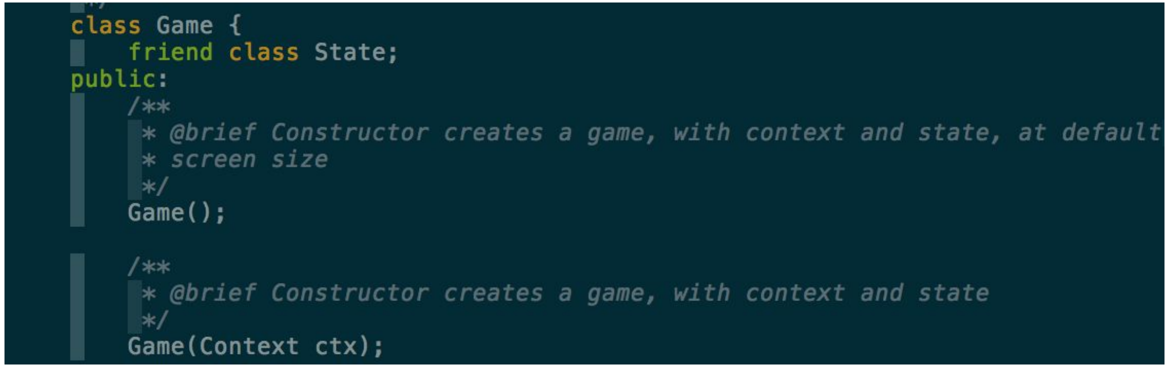
45
Unique cloners

Pong Walkthrough

Hope you like paddling

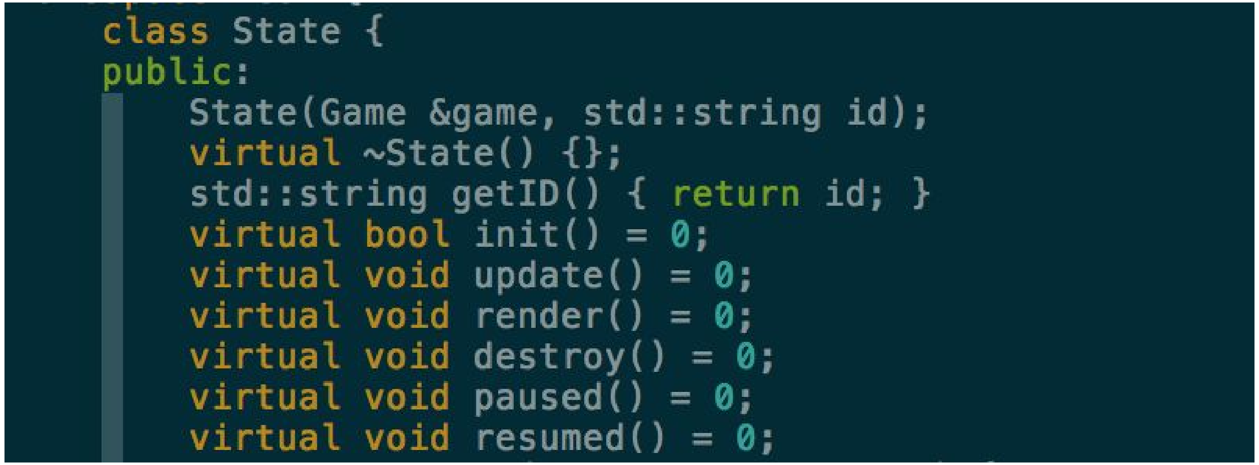
Creating the Game object

Game constructors



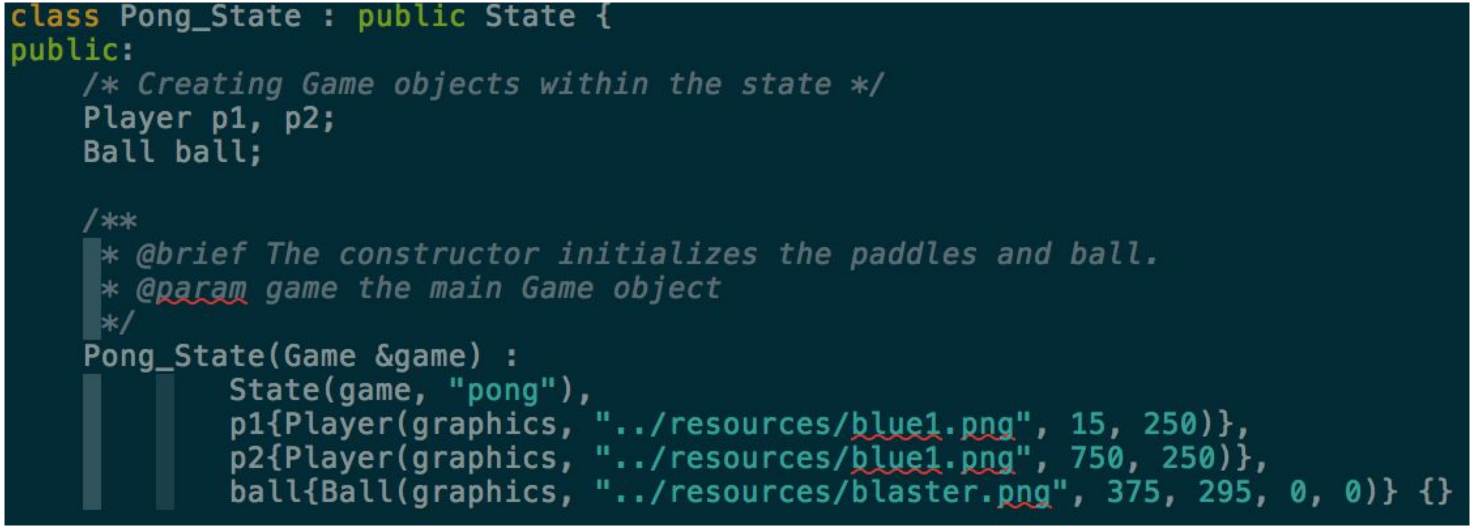
```
class Game {
    friend class State;
public:
    /**
     * @brief Constructor creates a game, with context and state, at default
     * screen size
     */
    Game();

    /**
     * @brief Constructor creates a game, with context and state
     */
    Game(Context ctx);
```

The State Object



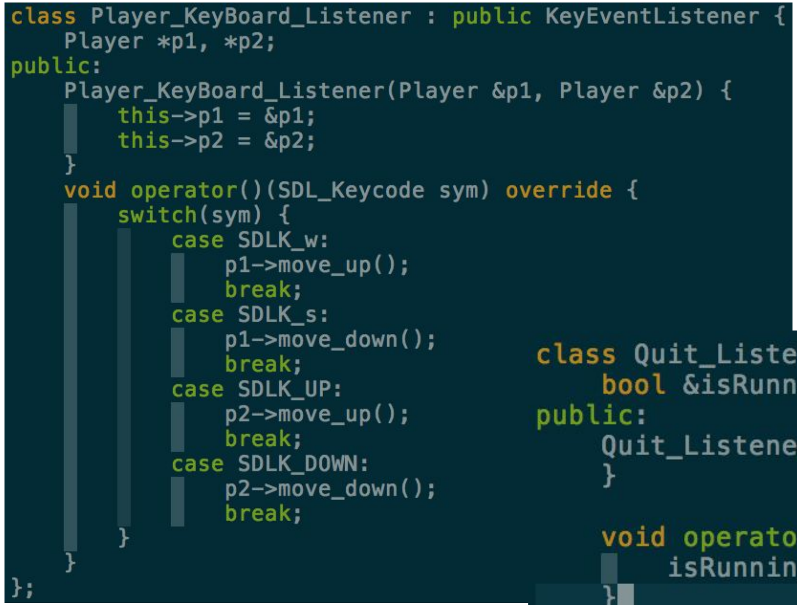
```
class State {
public:
    State(Game &game, std::string id);
    virtual ~State() {};
    std::string getID() { return id; }
    virtual bool init() = 0;
    virtual void update() = 0;
    virtual void render() = 0;
    virtual void destroy() = 0;
    virtual void paused() = 0;
    virtual void resumed() = 0;
```

The Pong_State Object

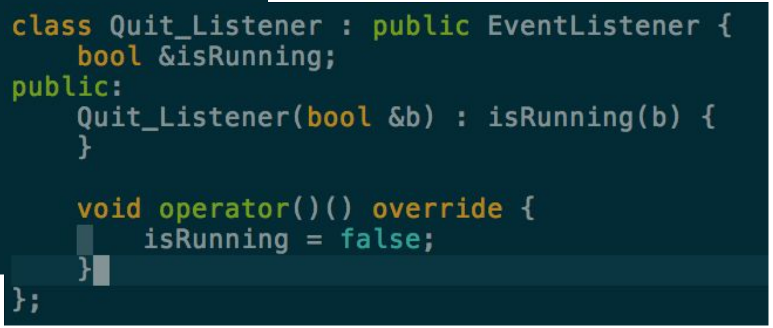


```
class Pong_State : public State {
public:
    /* Creating Game objects within the state */
    Player p1, p2;
    Ball ball;

    /**
     * @brief The constructor initializes the paddles and ball.
     * @param game the main Game object
     */
    Pong_State(Game &game) :
            State(game, "pong"),
            p1{Player(graphics, "../resources/blue1.png", 15, 250)},
            p2{Player(graphics, "../resources/blue1.png", 750, 250)},
            ball{Ball(graphics, "../resources/blaster.png", 375, 295, 0, 0)} {}
```

The Event listener

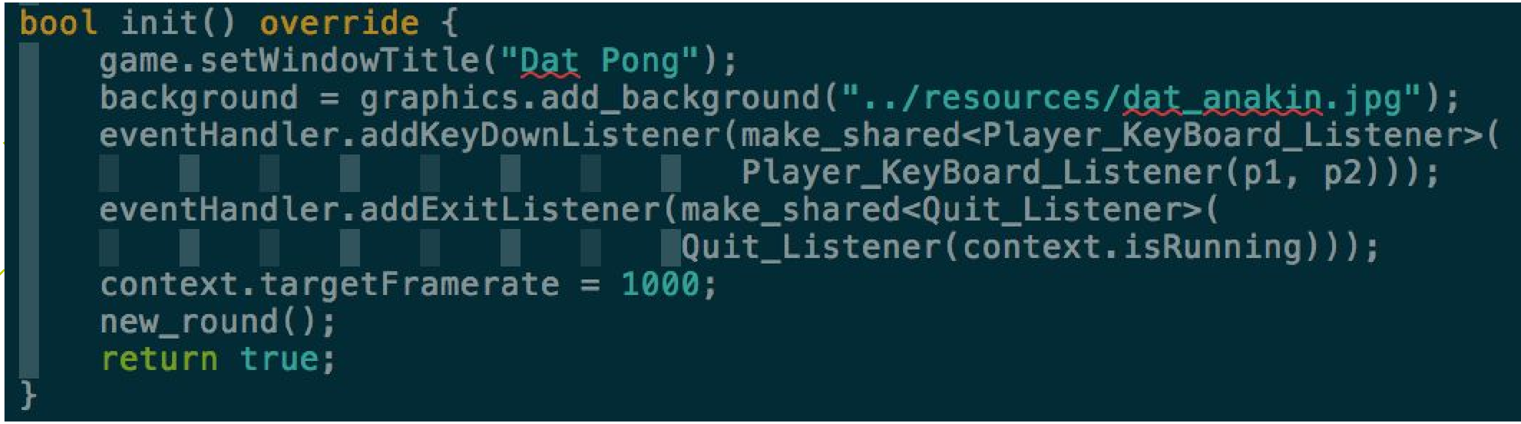


```
class Player_KeyBoard_Listener : public KeyEventListener {
    Player *p1, *p2;
public:
    Player_KeyBoard_Listener(Player &p1, Player &p2) {
        this->p1 = &p1;
        this->p2 = &p2;
    }
    void operator()(SDL_Keycode sym) override {
        switch(sym) {
            case SDLK_w:
                p1->move_up();
                break;
            case SDLK_s:
                p1->move_down();
                break;
            case SDLK_UP:
                p2->move_up();
                break;
            case SDLK_DOWN:
                p2->move_down();
                break;
        }
    }
};
```



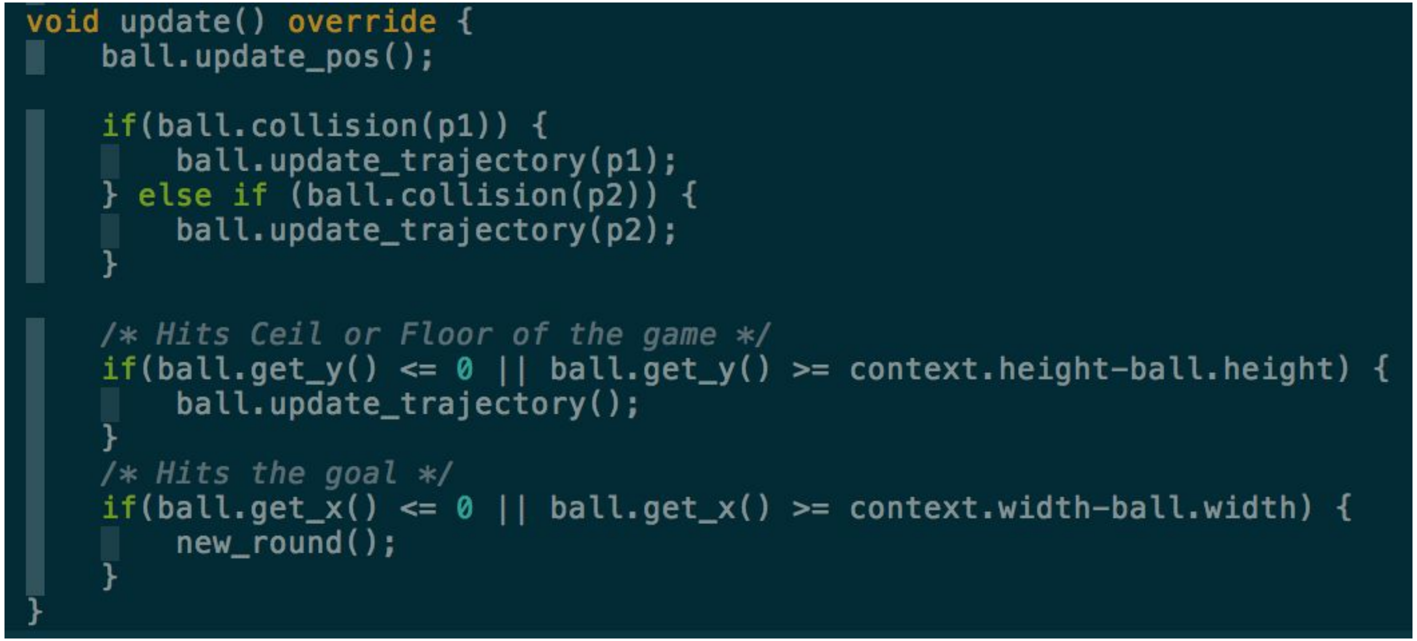
```
class Quit_Listener : public EventListener {
    bool &isRunning;
public:
    Quit_Listener(bool &b) : isRunning(b) {
    }

    void operator()() override {
        isRunning = false;
    }
};
```

The Pong_State Init



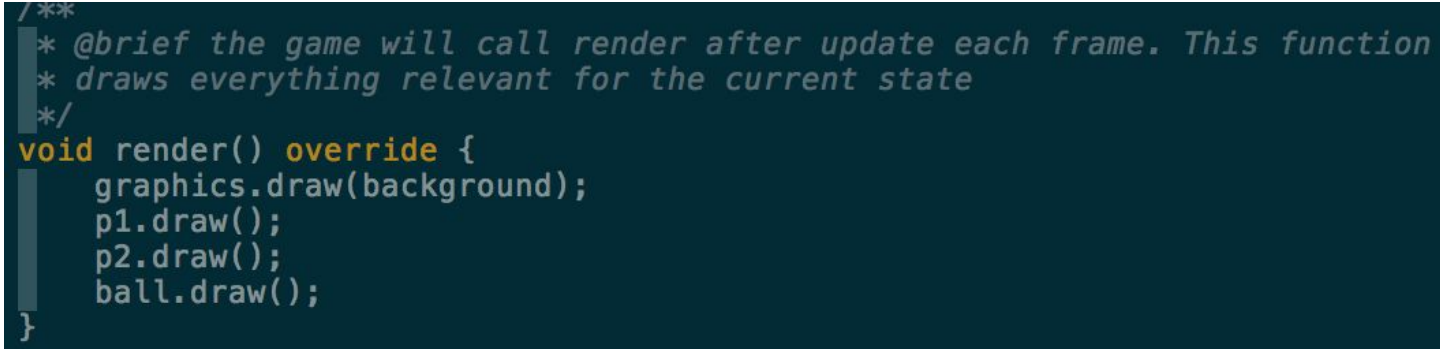
```
bool init() override {
    game.setWindowTitle("Dat Pong");
    background = graphics.add_background("../resources/dat_anakin.jpg");
    eventHandler.addKeyDownListener(make_shared<Player_KeyBoard_Listener>(
                                        Player_KeyBoard_Listener(p1, p2)));
    eventHandler.addExitListener(make_shared<Quit_Listener>(
                                    Quit_Listener(context.isRunning)));
    context.targetFramerate = 1000;
    new_round();
    return true;
}
```

The Pong_State Update



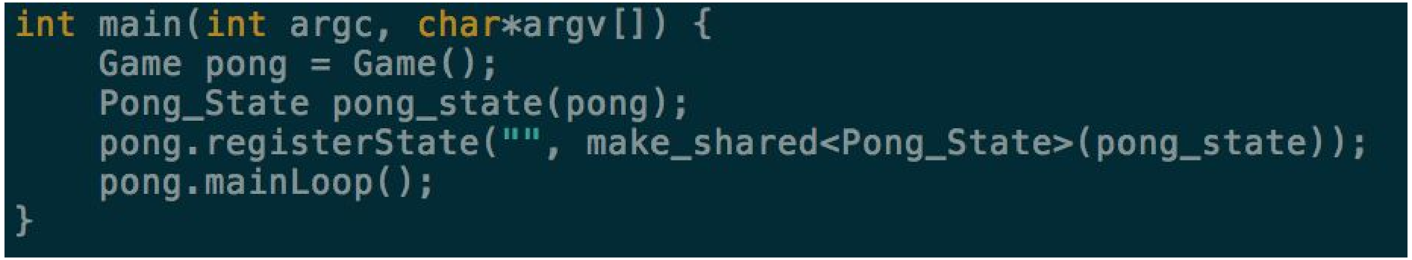
```
void update() override {
    ball.update_pos();

    if(ball.collision(p1)) {
        ball.update_trajectory(p1);
    } else if (ball.collision(p2)) {
        ball.update_trajectory(p2);
    }

    /* Hits Ceil or Floor of the game */
    if(ball.get_y() <= 0 || ball.get_y() >= context.height-ball.height) {
        ball.update_trajectory();
    }
    /* Hits the goal */
    if(ball.get_x() <= 0 || ball.get_x() >= context.width-ball.width) {
        new_round();
    }
}
```

The Pong_State Render



```
/**
 * @brief the game will call render after update each frame. This function
 * draws everything relevant for the current state
 */
void render() override {
    graphics.draw(background);
    p1.draw();
    p2.draw();
    ball.draw();
}
```

Pong Putting it all together



```
int main(int argc, char*argv[]) {
    Game pong = Game();
    Pong_State pong_state(pong);
    pong.registerState("", make_shared<Pong_State>(pong_state));
    pong.mainLoop();
}
```

Dat Pong

Testing Strategy

Getting testy up in here

Units Tests in Game Libraries

- Game libraries != normal libraries
- Demos win
- Need lots of demos to affirm

Conclusions

Work in Progress

Future Work

- Need to add more interfaces
- Hide more of SDL from user
- Create more example games
- Add multithreading

Future Work

- Add sound
- Add drawing shapes
- Add internet plugins
- Concepts support

Lessons Learned

- Get started early! (not 8am today)
- Game libraries need to be well thought out
- C++ is very useful