Platformer 2D

Computer Programming 4

Szymon Pluta

Informatics
Instructor: mgr inż. Wojciech Dudzik
Submit date: 19.06. 2020

Faculty of Automatic Control, Electronics and
Computer Science
Silesian University of Technology

Contents

1 Topic

The topic of presented project is a customizable Platformer 2D game.

The game centres the focus not only on the main game loop logic, but also on various features in order to build a whole enjoyable game experience.

The game has possibly decoupled independent modules, which can be easily re-used in different games as pieces of a game engine.

Finally, the game is easily customizable and is opened for future development extensions outside of this report and course scope.

2 Topic Analysis

In order to provide an enjoyable general gaming experience:

2.1 Requirements

The following requirements were assumed and met by the project.

2.1.1 The game logic

The world is possibly a huge tile-map (e.g. 500×500 entities, which is 16000×16000 px assuming entity is 32×32 px).

The world should be filled mostly with the collidable block entity obstacles. The goal for the player is to find the **Objective** in such a vast, customizable world. Map creators can add a considered number of **Spikes** or **Hearts** on the players' path to reaching the verb—objective—.

Player can collect `heartsto increase life, and die if touch a \verb`spike or fall into void (below the map's edge).

The game difficulty can be tuned in by adjusting:

- map size
- number of encountered `spikes` and `hearts`
- making clever dead-ends, overall blocks density, etc.
- physics e.g. gravity or horizontal velocity.

The camera is fixed on a player and the gameplay works smoothly.

There is an extra Hearts overlay and Fps overlay showing during the main game.

2.1.2 World logic

A world is generated from external files. I have chosen `.bmp` and `.txt` as the files formats.

The `.bmp` map files can be easily created within graphical softwares such as GIMP or MS Paint, hence new maps can be created very quickly. It is also easy to modify such maps, given the fact the game world is filled with regular square entities.

The `.txt` is an alternative to `.bmp`. The txt maps can be created: **manually**, within my separate dedicated map editor project, or with a different external software. In the end, a dedicated map editor will be even more convenient than a GIMP, and introduces a lot of possible extensions to the whole game ecosystem.

2.1.3 Game Menus

I am seeing more and more games implementing only the main game loop, and possibly a menu. My game provides a lot of various different game states:

- StateGame
- StateMenu
- StateAbout
- StateMapLoader
- StateOptions
- StatePaused
- StateRestart
- StateKeybinds

Having a lot of states increase the game experience, and makes the players feel like the game is "more completed".

The menus are very **responsive** to events, it should never lag or glitch.

Lastly, the menus should look decent even though a simple design is chosen.

Navigation between states is straight forward.

2.1.4 Resources

The game should introduces a lot of different resources i.e. textures, sounds, fonts, music, GUI components to make up a better gameplay.

The resources is something that can be easily tuned if necessary. It is not difficult to add e.g. 100 different new Block textures to the game. It'd take time, however.

The resources are re-pluggable: player can plug in a different resource into his working directory (e.g. use different sounds).

2.1.5 Misc

Aa game must be stable

- the framerate should not vary much, change suddenly
- framerate independent movement
- it should not crash unexpectedly, preferably never
- it should be consistent within the player's imagination expectations e.g. a jump should be predictable, it must not be sometimes 10 units high, and sometimes 1000.
- it should be free of bugs affecting the gameplay at the very least, e.g. bugs-free collisions
- it should not break upon invalid input

2.1.6 Configuration

Aside from customizable maps and resources, the game should allow the player to alter the behaviour of the game.

Currently, there are tons of constexpr configurable variables enclosed within `config` namespace. Player can edit them in one place and re-compile. They can be moved to e.g. `.yaml` file.

I have added highly customizable options:

- Checkbox to toggle FPS display enable
- Checkbox to toggle Sound enable
- Slider to alter Sound volume
- Whole keybinds state allow the player to rebind his `jump`, `runLeft` and `runRight` actions.

The backgrounds and fonts are also configurable.

2.2 Data structures

The project uses various STL containers. I did not re-invent data structures.

Quad-tree data structure is occasionally used in game development, which could be something interesting to implement, but it was not necessary in this game.

I have implement my own containers, though.

Such an example is e.g. `ResourceHolder<Key, Resource>` that can find generic resource to fill in the requirements of having different resource types: e.g. sound and textures.

2.3 Algorithms

I have introduced my own collsion solving system for both the player and camera, and I believe they work very well. I described Camera algorithm in a great details within my doxygen documentation.

I also introduced my own `.bmp` and `.txt` reading algorithm, and world generation out of thereof.

2.4 Libraries decision

I have decided to use `SFML` library, partly because I am familiar with its documentation, and partly because it is a good fit for writing simple games.

However, `SFML` was missing a decent graphical user interface view layer (making buttons and such). Therefore I have also included `tgui` library, which is a library partially deriving from `SFML`, and hence both are easily intergrable.

The standard C++ library is used heavily, with modern `C++17` and `C++20` code as well.

I have also integrated `googletest` library with the project, especially for the future development needs.

There is one call to `WinAPI` function, but only if compiling on Windows, so the program is cross-platform anyway.

Future upgrades would include e.g.

- `yaml` parser library, so that the configuration variables can be easily factored out to a separate file
- `ctre` regex library, because it's compile time and the syntax is very clean
- networking library, so that the game can interact with map editor

As shown, this project is still eagerly opened for future extensions and ideas, even though it is already reasonable in its size.

3 External specification

The project is fully cross-platform.

3.1 Installation

Full installation guide for `MinGW` on Windows:

```
git clone https://github.com/Wenox/platformer-2d.git
cd platformer-2d
mkdir lib && mkdir test/lib && mkdir build
<Install SFML into lib/>
<Install tgui into lib/>
<Install dynamic SFML and tgui libraries into lib/shared_lib or build/>
<Install googletest into test/lib/>
cd build
cmake .. -G "MinGW Makefiles"
mingw32-make
./platformer-2d
```

For a more customizable installation consult with your compiler.

3.2 Map loader requirements

A requirement for a `.bmp` file is that its width is a multiple of 4 e.g. $44 or 480$ but not 45.

A requirement for a `.txt` file is that it is well-formatted (white-space separatted columns).

The encoded pixel colors for `.bmp` and integer values for `.txt` can be found in a separate `resources/Encoded_Objects.txt/"` file or directly inside `Encoder::encodeAll()` class definition.

Note: `.bmp` file format uses BGR color format, not RGB.

3.3 Game screens

I will present relevant screens of the game states and explain the interconnection between the states from a player's perspective.

3.3.1 State Game

The state where the main game logic is done.

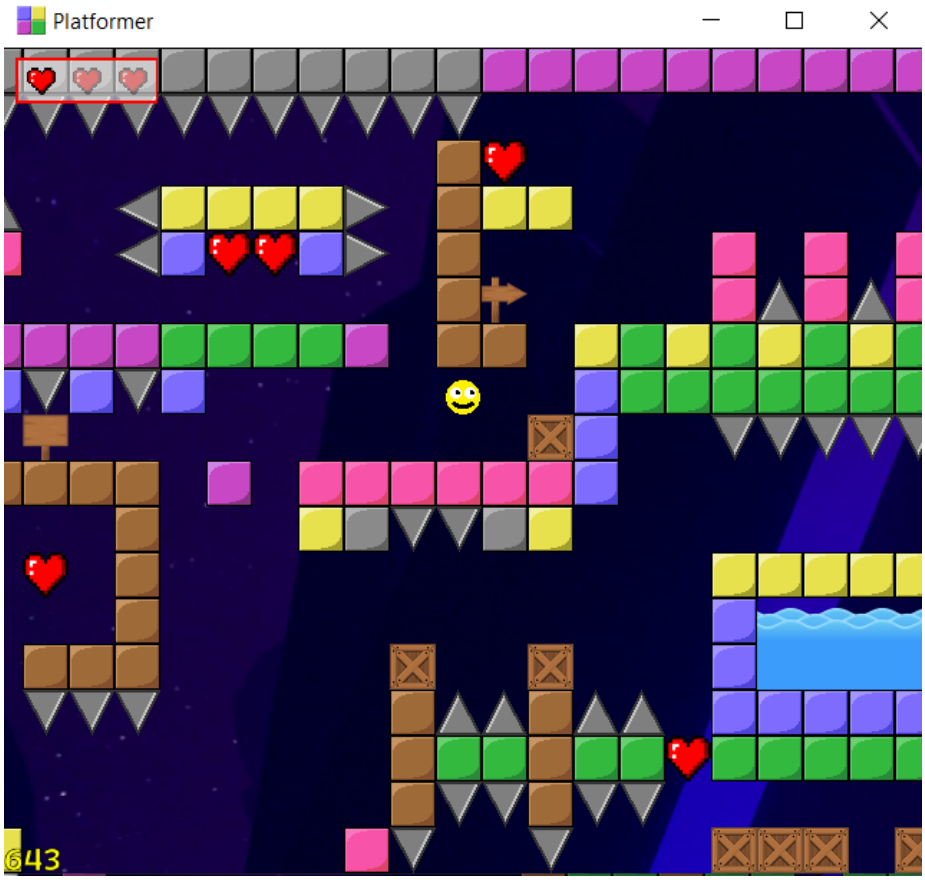


Figure 1: Gameplay screen

Available keybinds:

- `P` - `pause`
- `W` - `jump` - rebindable in Keybinds Menu
- `A` - `run left` - rebindable in Keybinds Menu
- `D` - `run right` - rebindable in Keybinds Menu

Player inside `StateGame` can switch to the following states:

- `StatePaused` - upon pause
- `StateRestart` - upon lost or won game
- `StateMenu` - upon exit

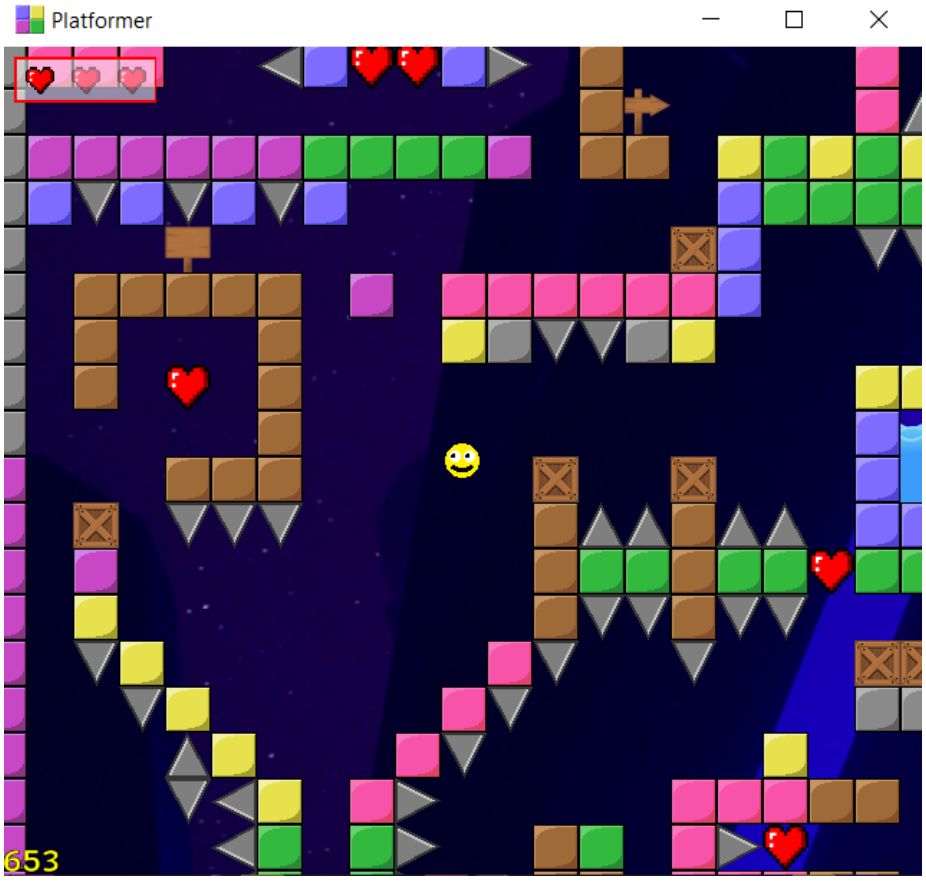


Figure 2: Gameplay Screen

Player lives overlay update upon death or regaining life.

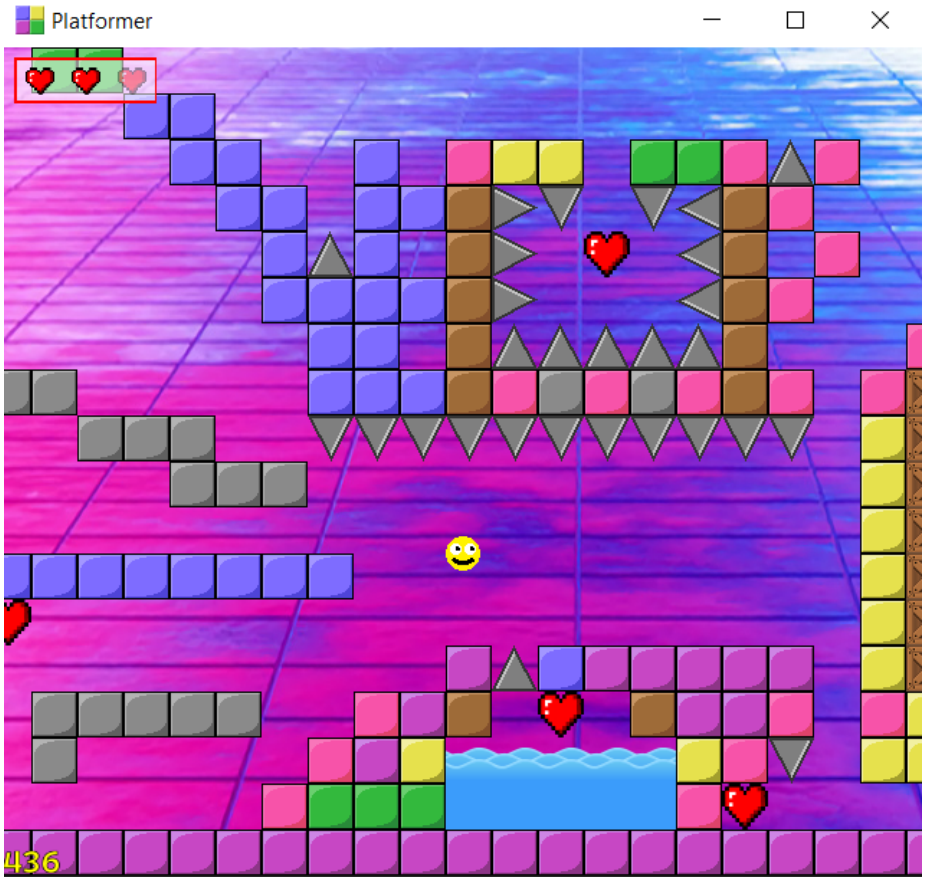


Figure 3: Player regained health

Player found the objective door.

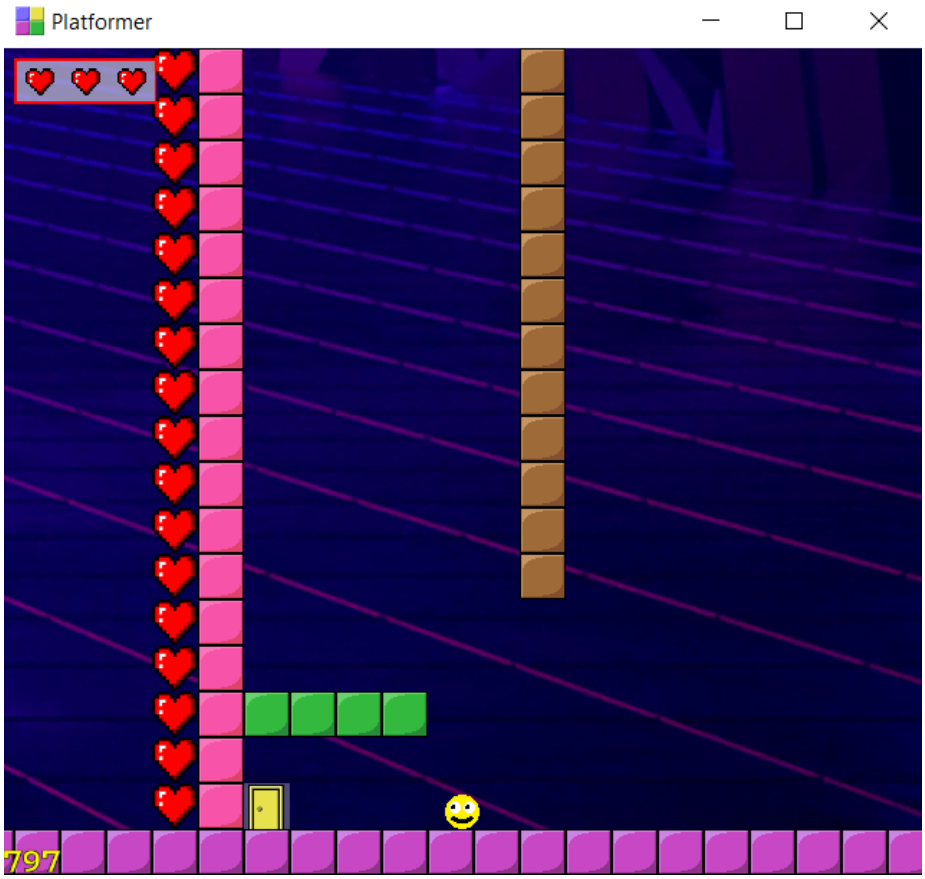


Figure 4: Objective

3.4 Game map

The above game screens were generated from currrent default map: `map.bmp`.

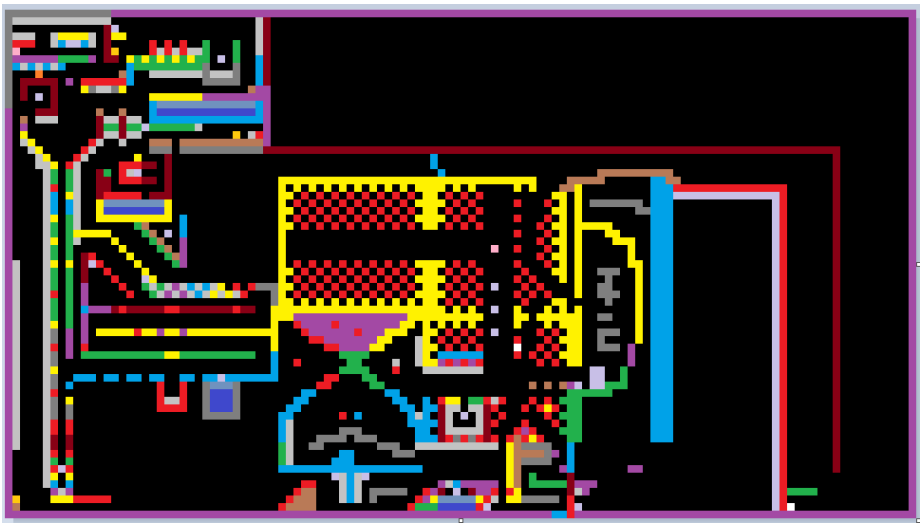


Figure 5: The default map, not for scale

The above map is of 120×67 pixels size.

For details see: my custom doxygen.

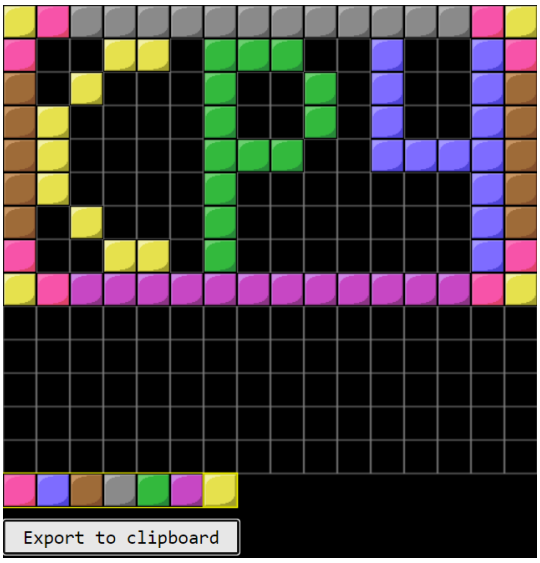


Figure 6: Separate map editor project made for this game.

3.5 State Paused

`StatePaused` is invoked only if player explicitly presses `P` button.

Pressing `ESC` within this state resumes the game.

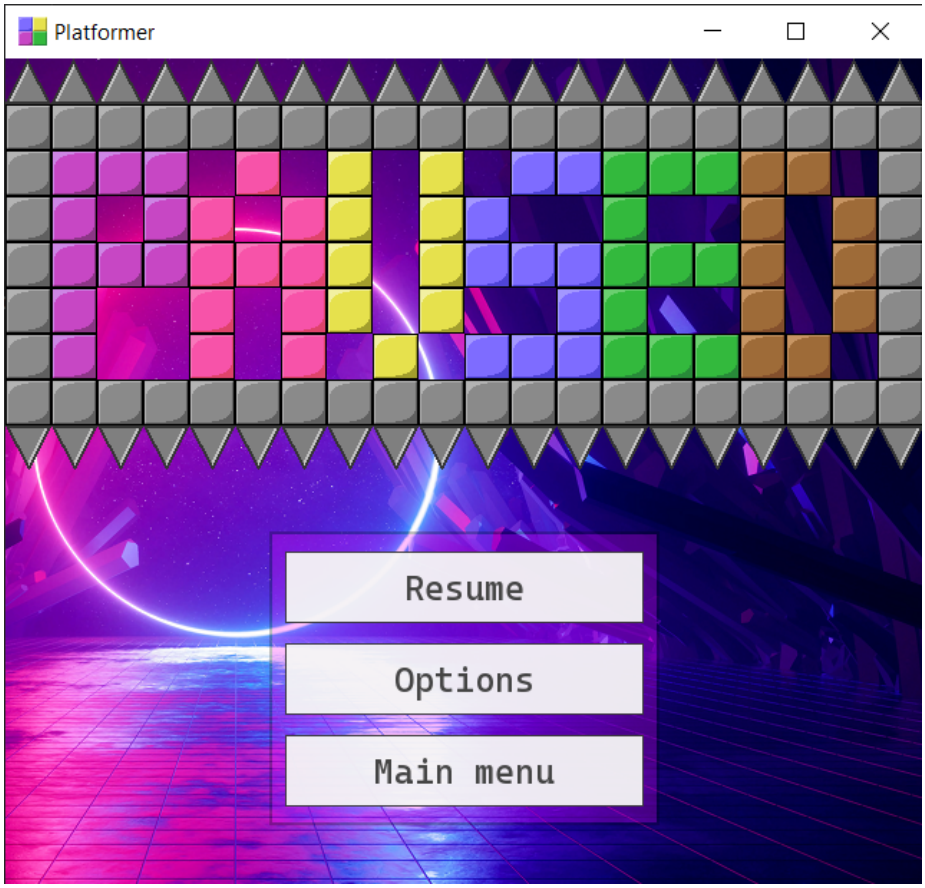


Figure 7: Paused menu

Player inside `StatePaused` can switch to the following states:

- `StateGame`
- `StateOptions`
- `StateMenu`

3.6 State Menu

`StateMenu` is the default state where the player starts the game.

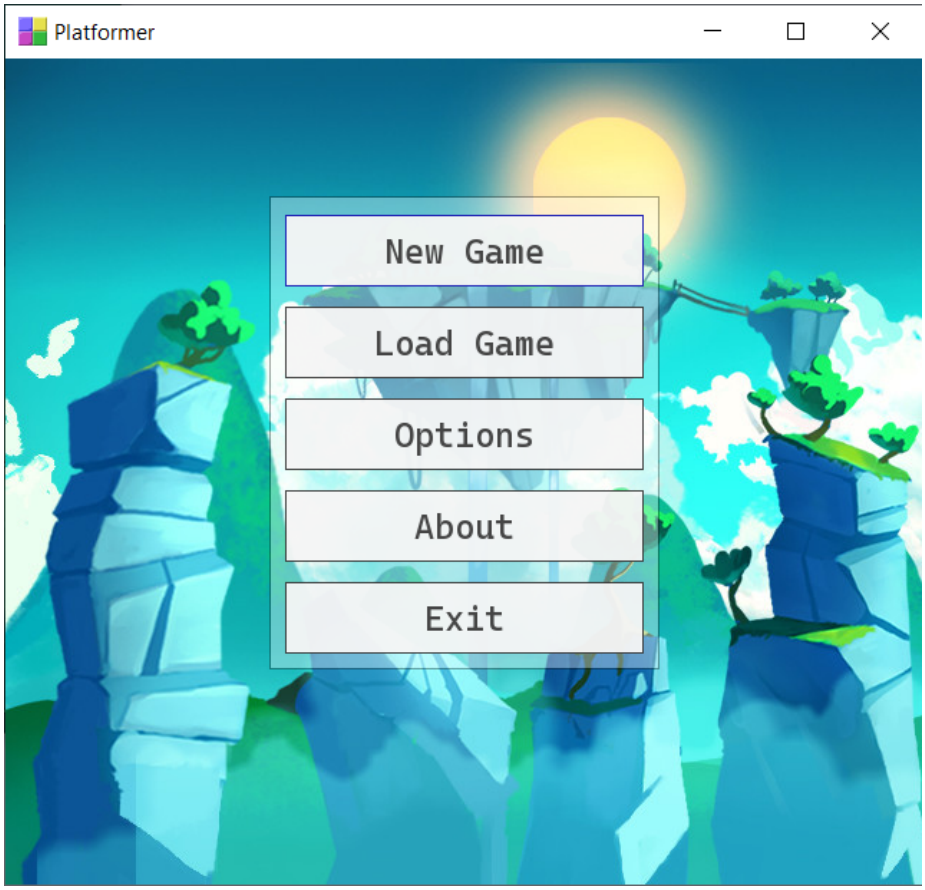


Figure 8: Main menu

Player inside `StatePaused` can switch to the following states:

- `StateGame` upon pressing `New Game` button, then the `StateGame` is constructed from the default `map.bmp`
- `StateMapLoader` by pressing `Load Game` button
- `StateOptions`
- `About`

Note: pressing the `Exit` button is the condition for closing the window, and hence end the main game loop.

3.7 State Map Loader

`StateMapLoader` is the state where player can load his custom `txt` or `bmp` maps.

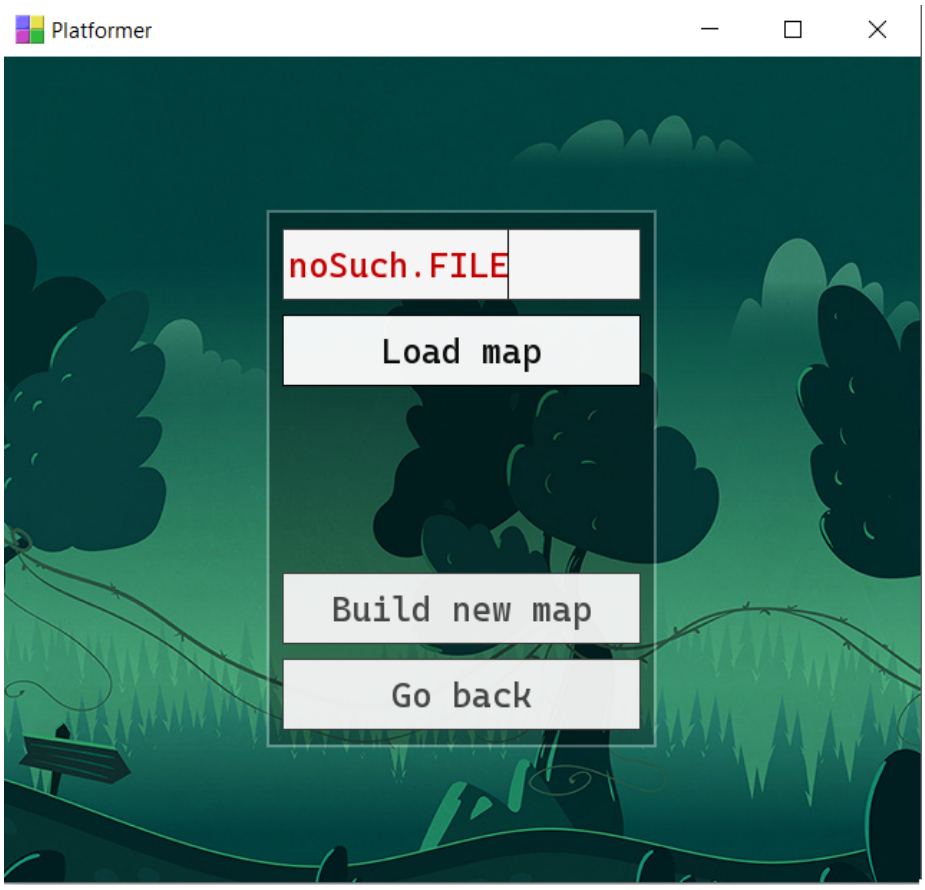


Figure 9: Trying to load an invalid map file *(see next screen for a response)*

Player inside `StateMapLoader` can switch to the following states:

- `StateGame` upon successful map creation constructs it
- `StateMenu` by going back

Response when bad map was entered:

- display `No such file!` with an animation
- prompt the user to `Enter map name...`
- the `Enter map name...` placeholder text nullifies when it is clicked, making a responsive feel

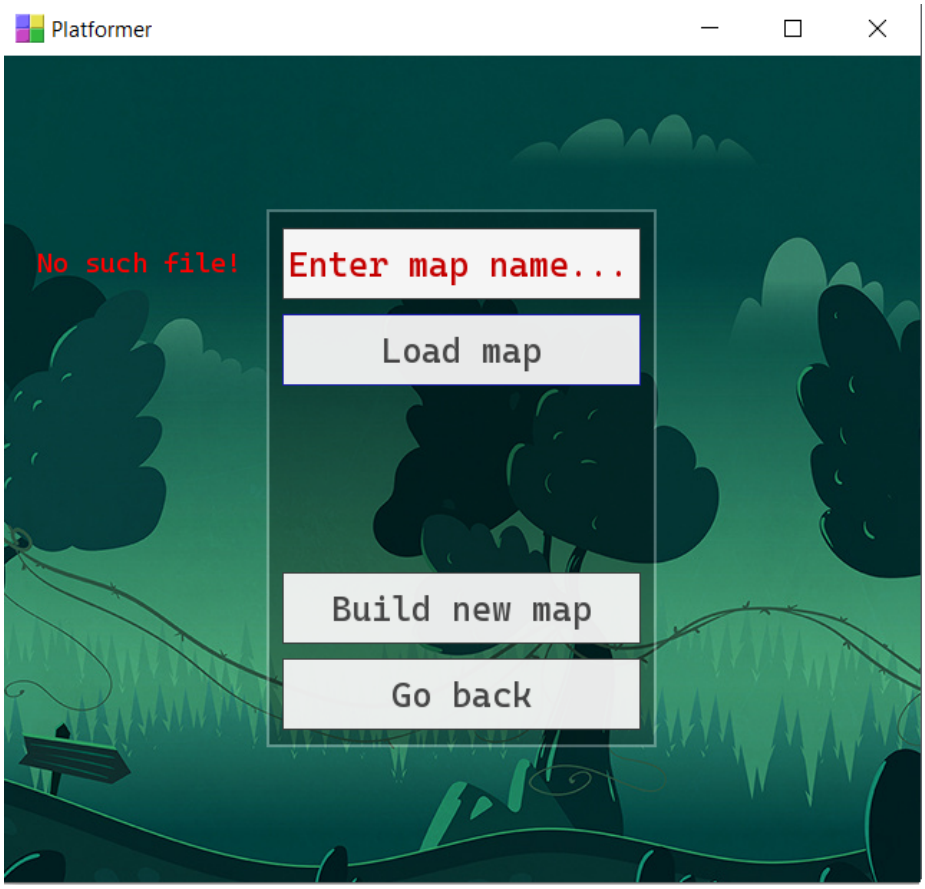


Figure 10: When invalid map name was entered, a proper response is made

Hovering over any button within my game `plays a sound` and slightly lights up the button.

3.8 State Options

`StateOptions` is the state where player can set up his run-time options.

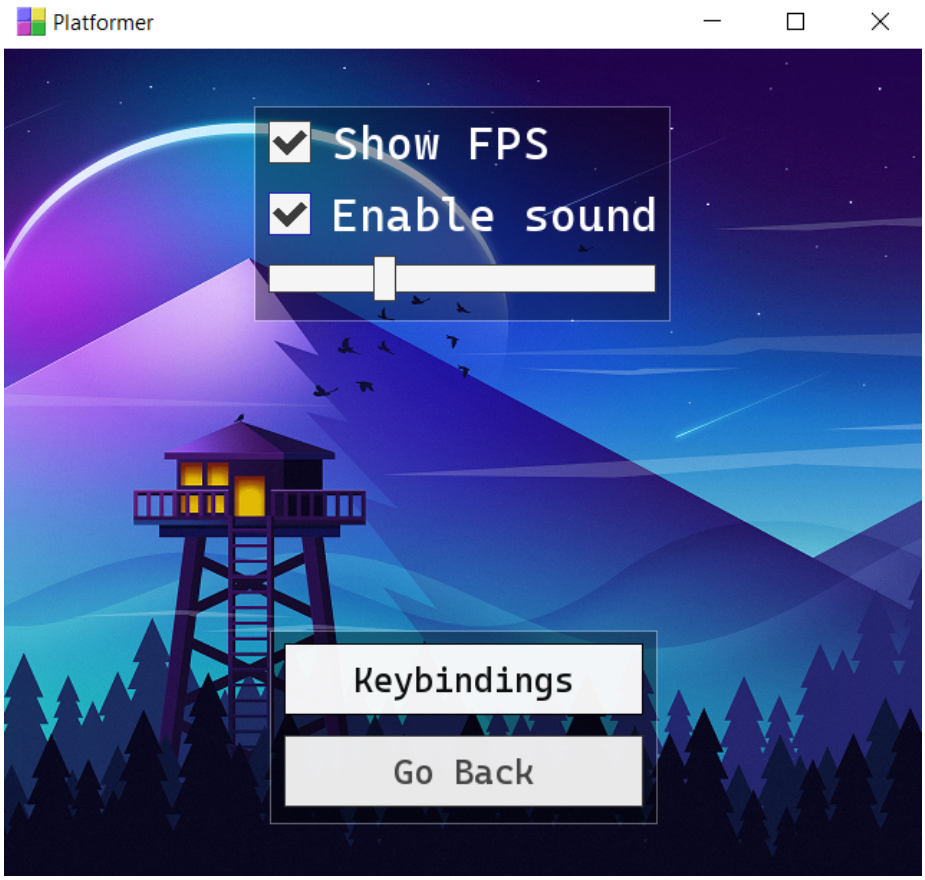


Figure 11: Options Menu

Player inside `StateOptions` can switch to the following states:

- `StateKeybinds` if `Keybinds` was pressed
- `StatePaused` if `Go Back` was pressed and player entered Options from there
- `StateMenu` if `Go Back` was pressed and player entered Options from there

The sound hover disables itself and lowers the opacity when sound checkbox is disabled.

Updating the sound volume options takes effect **immedietely**.

3.9 State Keybinds

`StateRebind` is the state where player can set up his run-time action keybindings.

Available actions that can be rebound:

- Run left - default keybind: `A`
- Run right - default keybind: `D`
- Jump - default keybind: `W`

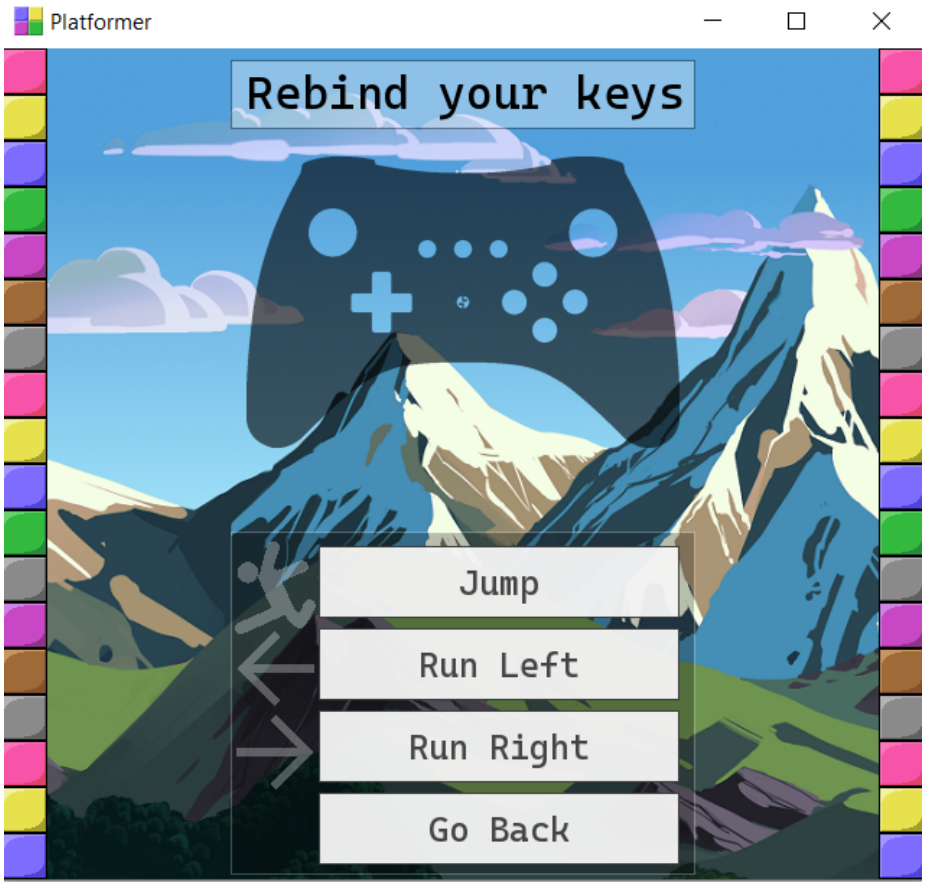


Figure 12: Keybinds Menu

Player inside `StateKeybinds` can switch to the following states:

- `StateOptions` if `Go Back` or `ESC` was pressed

The user is prompted to rebind his key with a large `Rebind your keys` label and large gamepad icon.

When any action button is pressed, then the gamepad smoothly fades away, and the below `Press key...` gui smoothly fades in.

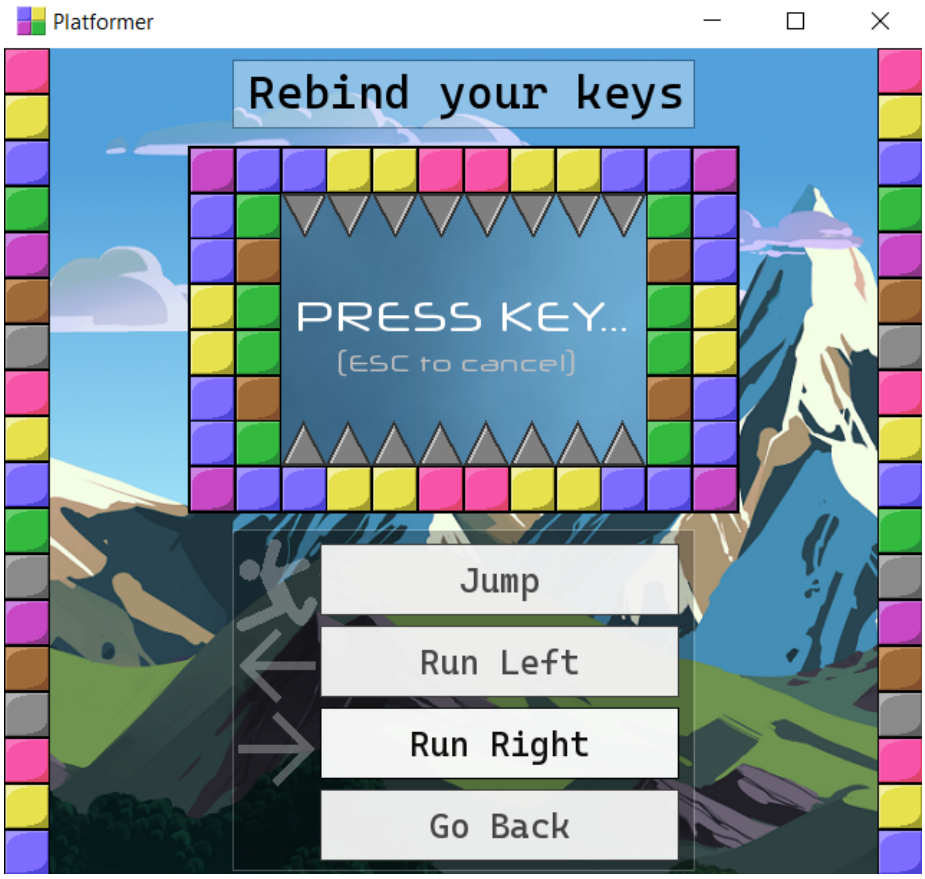


Figure 13: A keybind is being registered

The user can either press `ESC` to cancel rebinding, or press an arbitrary key to bind that given action to it.

The user is prompted to rebind his key with a large `Rebind your keys` label and large gamepad icon.

When any action button is pressed, then the gamepad smoothly fades away, and the below `Press key...` gui smoothly fades in.

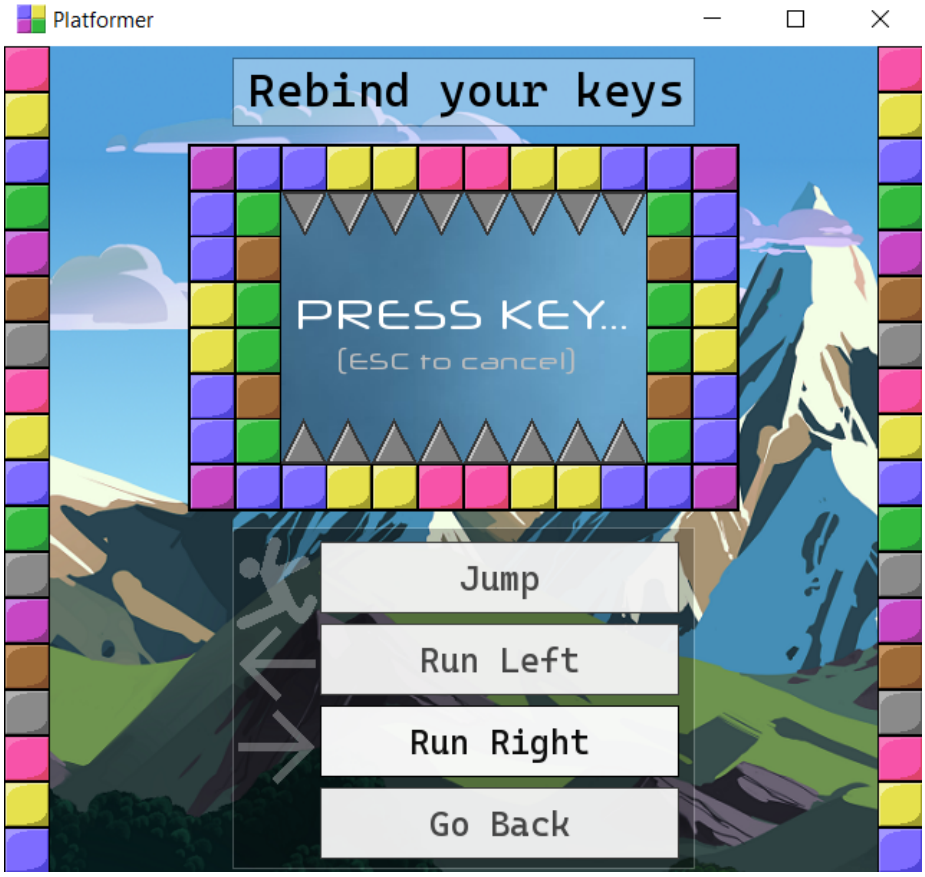


Figure 14: A keybind is being registered

3.10 State About

`StateAbout` prints a placeholder texture.

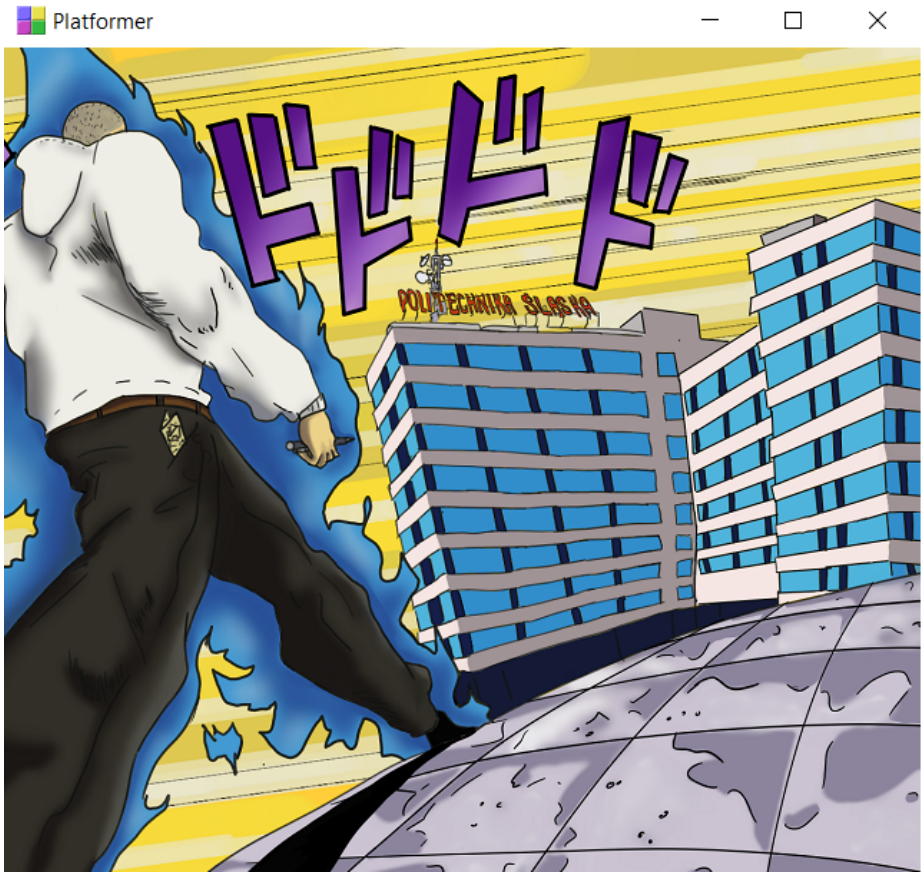


Figure 15: Keybinds Menu

Player inside `StateAbout` can switch to the following states:

- `StateMenu` `ESC` was pressed

4 Internal specification

I have commented every file with custom (not auto-generated), readable doxygen comments. The generated documentation is very well informative.

Doxygen online docs: **https://wenox.github.io/platformer-2d/html/modules.html**

My github: **https://github.com/wenox/platformer-2d**

I am also including Doxygen `.pdf` with $\approx$ 550 pages as a separate file.

I believe my commented doxygen documentation will be much more readable. Having 63 different `headers`, I will only mention here the most crucial functionalities.

5 Used techniques

5.1 RTTI

```
template <typename Resource>
class MissingResource : public std::exception {
    std::string msg{};

    static std::string initMsg(std::string_view fileName) {
        return "Failed loading resource: "
               "{ Type: "    + std::string(typeid(Resource).name()) + ", "
               "File name: " + fileName.data()                      + " }";
    }

public:
    MissingResource() = default;
    MissingResource(std::string_view fileName) : msg{initMsg(fileName)}
    {}

    const char* what() const noexcept override {
        return msg.data();
    }
```

`typeid(Resource).name()` increases error messages readibility. Example:

```
what(): Failed loading resource: { Type: N2sf7TextureE, File name: Heart_Empty.png }
```

5.2 Exceptions

Previously shown verb—MissingResource— function template is called within ResourceHolder::insert:

```
template <typename... Args>
void insert(const Key& key, std::string_view fileName, Args&&... args) {
    auto resPtr = std::make_unique<Resource>();

    bool loaded{};
    if constexpr (std::is_same<Resource, sf::Music>()) {
        loaded = resPtr->openFromFile(resourcesDir + fileName.data(), std::forward<Args>(args)...);
    } else {
        loaded = resPtr->loadFromFile(resourcesDir + fileName.data(), std::forward<Args>(args)...);
    }

    if (!loaded) {
        throw MissingResource<Resource>(fileName);
    }
    resources.emplace(key, std::move(resPtr));
}
```

Catching the error from `ResourceManager` constructor:

```
ResourceManager::ResourceManager() {
    try {
        loadResources();
    } catch (const std::exception& e) {
        std::cerr << e.what() << std::endl;
        std::exit(EXIT_FAILURE);
    }

}
```

Different custom Exception class:

```
class MissingFont : public std::exception {
    std::string msg;
public:
    MissingFont(std::string msg = "Missing font") : msg(std::move(msg)) {}

    const char* what() const noexcept override {
        return msg.c_str();
    }
};
```

Method inside `View` base class.

```
void createBackgroundFrom(std::string_view backgroundName) {
        try {
                view.add(tgui::Picture::create(backgroundName.data()));
        }
        catch (const std::exception& e) {
                std::cerr << "No such background: " << backgroundName << '\t'
                          << "Using default background instead'"      << '\n';
                std::cerr << e.what() << std::endl;
        }
}
```

There is much more exception handling e.g.:

Setting the `Window` icon inside `Window` constructor:

```
try {
    setWindowIcon();
} catch (const std::exception& e) {
    std::cerr << "Using default game icon\n";
    std::cerr << e.what() << std::endl;
}
```

```
void KeybindsView::createRebindingPanelGroup() {
    const auto& parent = createRebindingPanel();
    try {
        addBackgroundInto(parent);
    } catch (const std::exception& e) {
        std::cerr << "Add keybinds background failed, using empty background instead.\n"
        std::cerr << e.what() << std::endl;
    }

    try {
        addPressKeyLabelInto(parent);
        addCancelLabelInto(parent);
    } catch (const std::exception& e) {
        std::cerr << "Add keybinds register labels failed. No such font: "
                  << config::keybindsFontName << std::endl;
        std::cerr << e.what() << std::endl;
}
}
```

5.3 Templates

I have many different templates. I will list a few examples.

```
template <typename T>
class FileReader {
        public:

        virtual void readFile() = 0;

        auto& getData() {
                return data;
        }

        virtual ~FileReader() = default;
        FileReader(FileReader&&) noexcept = default;
        FileReader& operator=(FileReader&&) noexcept = default;


        auto& getData() const {
                return data;
        }

        bool isOpened() const {
                return file.is_open();
        }

        protected:
        explicit FileReader(const std::string& name, const std::ios_base::openmode& mode = s
        : fileName{name}
        , file{name, mode}
        {}

        std::string    fileName;
        std::ifstream  file;
        std::vector<T> data{};
};
```

Inheriting classes using this template base:

```
class BmpReader final : public FileReader<PixelColor> {}
class TxtReader final : public FileReader<int> {}
```

Custom `Mappable` concept (C++20):

```
#if (__cplusplus == 202002L)
    template <typename Key>
    concept Mappable = std::strict_weak_order<std::less<Key>, Key, Key>;
#endif
```

Template deduction guide:

```
template <typename T, typename... Args>
ResourceInserter(T, std::string_view, Args... args) -> ResourceInserter<T, Args...>;
```

Overload lambda 'hack' for `std::visit`:

```
template<class... Ts> struct overload : Ts... { using Ts::operator()...; };
template<class... Ts> overload(Ts...) -> overload<Ts...>;
```

Default template argument

```
template <typename TWidgetPtr = tgui::Widget::Ptr>
class View {
protected:
tgui::Gui view;
int buttonsCounter{0};

virtual void buildGUI() = 0;

public:
std::vector<TWidgetPtr> widgets;


virtual ~View() = default;
View(const View&) = default;
View(View&&) noexcept = default;
View& operator=(const View&) = default;
View& operator=(View&&) noexcept = default;
/** etc */
```

Static polymorphism:

```
template <typename ReaderKey>
#if (__cplusplus == 202002L)
requires Mappable<Key>
#endif
class Encoder {
        public:
        std::map<ReaderKey, Obj::Entity> encodedObjects;

        constexpr Encoder() {
                static_assert(std::is_same<ReaderKey, PixelColor>()
                or std::is_same<ReaderKey, int>());

                this->encodeAll();
        }

        private:
        void encodeAll() {
                if constexpr (std::is_same<ReaderKey, PixelColor>()) {
                        encode(PixelColor{0, 0, 0},       Obj::Entity::Empty);
                        encode(PixelColor{201, 174, 255}, Obj::Entity::Player);
                        encode(PixelColor{255, 255, 255}, Obj::Entity::Objective);
                        /// ... etc
                }
                if constexpr (std::is_same<ReaderKey, int>()) {
                        encode(0, Obj::Entity::Empty);
                        encode(1, Obj::Entity::Player);
                        encode(9, Obj::Entity::Objective);
                        /// ... etc
                }
        }

    void encode(const ReaderKey& key, Obj::Entity entity) {
        static_assert(std::is_same<typename std::decay<decltype(key)>::type , ReaderKey>());
        encodedObjects.insert(std::make_pair(key, entity));
}
```

Utility template methods (the latter two are a protopy of boost mapListOf)

```
template <typename E>
constexpr auto to_underlying(E e) noexcept {
    return static_cast<std::underlying_type_t<E>>(e);
}

template <typename E>
constexpr std::optional<int> toInt(E e) noexcept {
    if constexpr (std::is_enum<E>::value)
        return static_cast<int>(e);
    return std::nullopt;
}

template<typename T>
struct mapListOfHelper
{
    T& data;
    constexpr explicit mapListOfHelper(T& d) : data(d) {}
    constexpr mapListOfHelper& operator()(typename T::key_type    const& key,
                                          typename T::mapped_type const& value) {
        data[key] = value;
        return *this;
    }
};

template<typename T>
constexpr  mapListOfHelper<T> mapListOf(T& item) {
    return mapListOfHelper<T>(item);
}
```

5.4 STL Containers

I am using the following STL containers.

5.4.1 Queue usages

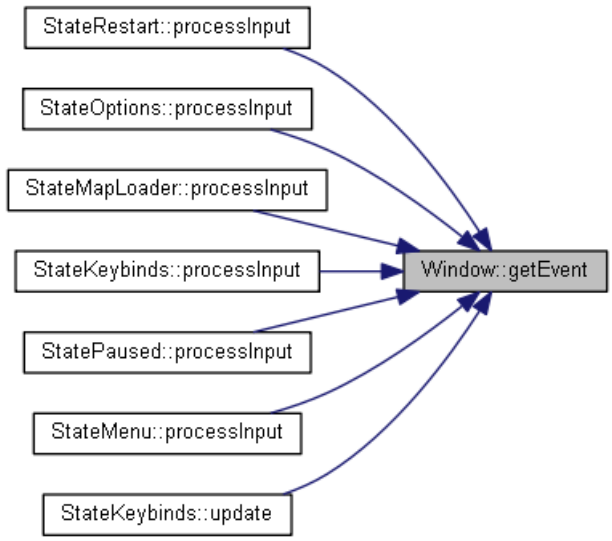
```
std::queue<res::Texture> queue;       ///< populated within MapLoader<FileType>

/// and then...

std::queue<res::Texture> blocksQueue; ///< private field within StateGame class

/** Passed to StateGame in its constructor: */

std::visit(overload{
    [&](MapLoader<Bmp>&) { blocksQueue = std::get<MapLoader<Bmp>>(mapLoader).getQueue(); },
    [&](MapLoader<Txt>&) { blocksQueue = std::get<MapLoader<Txt>>(mapLoader).getQueue(); },
}, mapLoader);
```

Those queues ensure that blocks are set with correct textures, according to the order they were presented in the input file map (txt or bmp).

And this queue:

```
std::queue<sf::Event> events;         ///< private field within Window class
```

Is retrieved retrieved every frame by all States but StateAbout.



`std::queue<sf::Event>& window::getEvent()`: Retrieve the events queue.

Called every frame by all classes having their own View gui instance.

Ensures correct drawing behaviour of View gui instances.

5.5 Array usages

Inside StateOptions.h:

```
std::array<std::reference_wrapper<sf::Texture>, 4> cornersTextures;
std::array<sf::Sprite, 4> corners;

// Initialization by a StateOptions constructor member initializer list:

StateOptions::StateOptions(StateMachine& sM, Window& window, ResourceManager& resources)
: cornersTextures{{resources.getTextures()[res::Texture::OptionsLeftTopCorner],
                   resources.getTextures()[res::Texture::OptionsLeftBotCorner],
                   resources.getTextures()[res::Texture::OptionsRightBotCorner],
                   resources.getTextures()[res::Texture::OptionsRightTopCorner]}}
, /** other members */
{
    for (std::size_t i{0u}; i < cornersTextures.size(); ++i) {
        corners.at(i).setTexture(cornersTextures.at(i).get());
    }
    corners.at(0).setPosition(0, 0);
    corners.at(1).setPosition(0, 512);
    corners.at(2).setPosition(576, 512);
    corners.at(3).setPosition(576, 0);
}
```

5.6 Vector usages

Vector is used very heavily in different classes. Note, that vector is especially importat in **game development** as it is a cache-friendly contiguous memory block.

5.7 Map usages

Map was shown earlier on with Mappable concept. I have usages of unorderedmap, too.

```
class ActionMap {
    std::unordered_map<std::string, sf::Keyboard::Key> ActionMap::actionMap;
    // ...
}
class InputEvent {
    std::unordered_map<std::string, sf::Keyboard::Key> InputEvent::keys;
    // ...
}


template <typename Key, typename Resource>
#if (__cplusplus == 202002L)
    requires Mappable<Key>
#endif
class ResourceHolder {
    std::unordered_map<Key, std::unique_ptr<Resource>> resources;
    // ...
}

class StateMachine {
    std::unordered_map<int, std::shared_ptr<State>> states{};
}

template <typename E>
struct IConfig {
    static_assert(std::is_enum<E>::value, "IConfig: E mustn't be Enum");

    std::map<E, const char*> widgetsNames;
}
```

5.8 STL Algorithms and iterators

I am using lambdas in std::variant, e.g.: Camera updateX(), and updateY() symmetrically.

```
/** Updating one axis at a time simplifies everything.
 *
 *  Upon entering updateX(), we know if in previous frame within updateY()
 *  method there was a Top, Bot or None collision, because we started it in previouslySolve
 *  container.
 *
 *  Similiarly, upon exiting updateX(), we store Left, Right or None inside previouslySolve
 *  depending on collision type. */

void Camera::updateX() noexcept {
    float solvedX = detectCollisionX();

    std::visit(overload{
        [&](Bot&)   { cameraView.setCenter(solvedX, bottomBorderBoundary); },
        [&](Top&)   { cameraView.setCenter(solvedX, topBorderBoundary); },
        [&](None&)  { cameraView.setCenter(solvedX, controller->getPosition().y + halvedSpri
        [&](auto) noexcept { }
    }, previouslySolvedOnOtherAxis);

    storeJustSolvedForOtherAxis(solvedNow);
}
```

Inside MapLoader::load():

```
void load() override {
    std::visit(overload{
        [&](BmpReader&) { analyzeBmpMapData(); },
        [&](TxtReader&) { analyzeTxtMapData(); },
        [&](std::monostate&) { }
    }, mapReader);
}
```

However, the most often used lambdas are those that bind a signal to gui widget.
Example:

```
void StateOptions::onCreate() {
        gui.widgets[to_underlying(Options::Btn::Keybinds)]->connect("Pressed", [&]() {
                stateMachine = state::keybindsID;
        });

        gui.widgets[to_underlying(Options::Btn::GoBack)]->connect("Pressed", [&]() {
                stateMachine.switchToPreviousState();
        });

        gui.getView().get("fpsCheckBox")->connect("Changed", [&]() {
                audioConfig.isFpsEnabled = gui.isFpsChecked();
        });

        gui.getView().get("soundCheckBox")->connect("Changed", [&]() {
                saveOptions();
                updateHoverSoundVolume();
                updateSlider();
        });

        gui.getView().get("soundVolume")->connect("ValueChanged", [&]() {
                audioConfig.volume = gui.getVolume();
                updateHoverSoundVolume();
        });

        for (auto& widget : gui.widgets) {
                widget->connect("MouseEntered", [&]() {
                        onHoverBtnSound.play();
                });
        }
}
```

Iterators find usage when calculating the number of columns in a txt file:

```
    config::blocksCountWidth  = std::distance(std::istream_iterator<int>{ss}, std::istream_i
```

The iteators are also used heavily within e.g. StateMachine methods, for example:

```
int StateMachine::insert(const std::shared_ptr<State>& state) {
    auto it = states.insert(std::make_pair(currentInsertedID, state));
    it.first->second->onCreate();

    return currentInsertedID++;
}
```

Lambdas and std::function are also used when solving Player¡-¿Collidable Blocks collisions.

```
void CollisionEvent::updateAxisX(float) {
        for (auto& block : blocks) {
                handleCollision(*block, [this](const Entity &e) {
                        this->resolveCollisionAxisX(e);
                });
        }
}

void CollisionEvent::updateAxisY(float) {
        for (auto& block : blocks) {
                handleCollision(*block, [this](const Entity &e) {
                        this->resolveCollisionAxisY(e);
                });
        }
}

void CollisionEvent::handleCollision(const Entity& block, const std::function<void(const Ent
        if (playerInRangeOf(block) && playerIntersects(block)) {
                resolveCollision(block);
        }
}
```

5.9 Smart pointers

Used heavily within all View classes, because `tgui::Widget::Ptr` is just a typedef to a shared pointer.

Example from KeybindsView header:

```
tgui::Panel::Ptr createRebindingPanel();
static void addBackgroundInto(const tgui::Panel::Ptr& parent);
static void addPressKeyLabelInto(const tgui::Panel::Ptr& parent);
static void addCancelLabelInto(const tgui::Panel::Ptr& parent);
```

Unique Pointers are used to be responsible for owning fundamental game entities inside GameState:

```
/**
std::vector<std::unique_ptr<Entity>> blocks;
std::vector<std::unique_ptr<Spike>> spikes;
std::vector<std::unique_ptr<HeartCollectible>> hearts;
*/
```

Events inside StateGame are also unique pointers:

```
td::unique_ptr<MovementEvent>  movementsEvent;
std::unique_ptr<CollisionEvent> collisionEvent;
std::unique_ptr<InputEvent>     inputEvent;

/** Inside onCreate() method: */
movementsEvent = std::make_unique<MovementEvent>(player, blocks);
collisionEvent = std::make_unique<CollisionEvent>(player, blocks);
inputEvent     = std::make_unique<InputEvent>(player, resources, window);
```

Lastly, all states themselves are shared pointers. Allocated at the very top level, except for StateGame which is allocated later.

```
Game::Game()
    : window{"Platformer"}
{
    this->init();
}

void Game::init() {
state::menuID     = stateMachine.insert(std::make_shared<StateMenu>     (stateMachine, window, resources));
state::loaderID   = stateMachine.insert(std::make_shared<StateMapLoader>(stateMachine, window, resources));
state::optionsID  = stateMachine.insert(std::make_shared<StateOptions>  (stateMachine, window, resources));
state::keybindsID = stateMachine.insert(std::make_shared<StateKeybinds> (stateMachine, window, resources));
state::pausedID   = stateMachine.insert(std::make_shared<StatePaused>   (stateMachine, window, resources));
state::restartID  = stateMachine.insert(std::make_shared<StateRestart>  (stateMachine, window, resources));
state::aboutID    = stateMachine.insert(std::make_shared<StateAbout>    (stateMachine, resources));

stateMachine.switchTo(state::menuID);
}
```

And even ResourceHolder template holds a unique pointer to a template Resource, as shown earlier.

For detailed classes info and best quality diagrams: https://wenox.github.io/platformer-2d

5.10 Threads

Aside from sounds that fire in a new threads, the only place where I use threads is:

```
ResourceManager::ResourceManager() {
        try {
                loadResources();
        } catch (const std::exception& e) {
                std::cerr << e.what() << std::endl;
                std::exit(EXIT_FAILURE);
        }
}

void ResourceManager::loadResources() {
        std::thread t1(&ResourceManager::loadTextures, this);
        std::thread t2(&ResourceManager::loadSounds, this);
        std::thread t3(&ResourceManager::loadMusic, this);
        t1.join();
        t2.join();
        t3.join();
}
```

5.11 Regular expressions

Regular expressions are used in two places:

Checking if a Txt file is a well-formed Txt map file:

```
bool TxtReader::isValidTxt() {
    /** Allocate the memory for raw txt data */
    file.seekg(0, std::ios::end);
    rawTxtData.reserve(file.tellg());
    file.seekg(0, std::ios::beg);

    /** Entire file is loaded into string */
    rawTxtData.assign((std::istreambuf_iterator<char>(file)),
    std::istreambuf_iterator<char>());

    /** And checked if it indeed is a valid Txt file */
    std::string fullMatchPattern = R"((?:[[:space:]]*[[:digit:]]+[[:space:]]*)+)";
    std::regex re{fullMatchPattern};
    if (!std::regex_match(rawTxtData, re)) {
        std::cerr << "File " << fileName << " does not match the pattern!\n";
        return false;
    }
        return true;
}
```

```cpp
/** Checking if a given fileName is a valid map name (ends with bmp or txt) */
class MapNameValidator {
    public:
    explicit MapNameValidator(std::string_view fileName);

    bool isValidFormat();
    bool exists() const();
    bool isBmp() const;
    bool isTxt() const;

private:
    const std::string_view fileName;

    std::regex re{};
    constexpr static std::string_view mapNamePattern = R"(.+?\.(bmp|txt))";

    std::match_results<std::string_view::const_iterator> match{};
};

MapNameValidator::MapNameValidator(std::string_view fileName)
: fileName{fileName}
{
    try {
        re.assign(mapNamePattern.data());
    } catch (const std::regex_error& e) {
        std::cerr << "Bad pattern: " << mapNamePattern << std::endl;
        std::cerr << e.what() << std::endl;
    }
}

bool MapNameValidator::isValidFormat() {
    return std::regex_match(fileName.data(), match, re);
}

bool MapNameValidator::exists() const {
    std::ifstream file{fileName.data()};
    return file.good();
}

bool MapNameValidator::isBmp() const {
    return match.str(1) == std::string("bmp");
}

bool MapNameValidator::isTxt() const {
    return match.str(1) == std::string("txt");
}
```

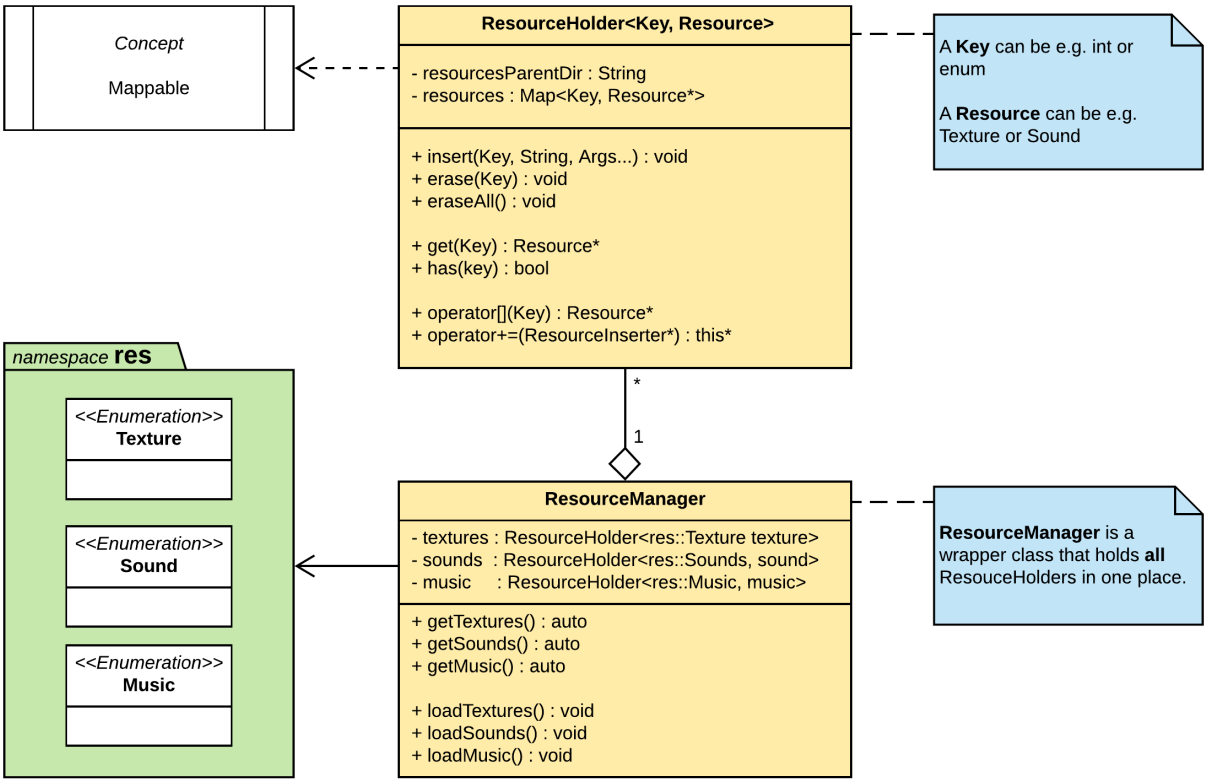
5.12 Relevant diagrams

Those are the diagrams I made for the presentation and are still relevant.

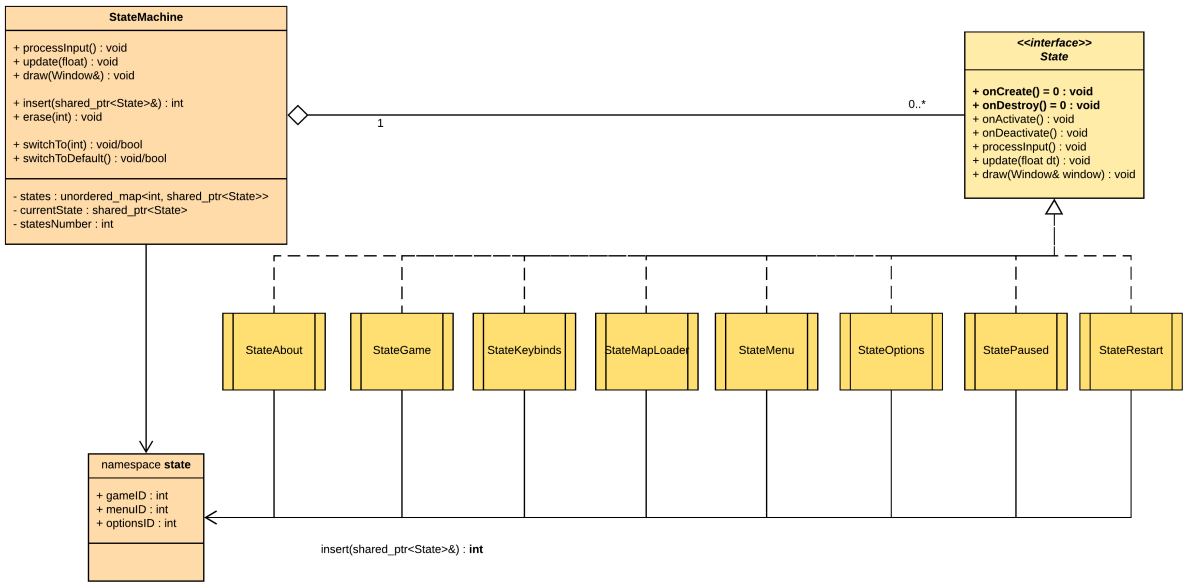
5.13 Resource management

All possible diagrams are included within both doxygen.pdf and doxygen github pages.

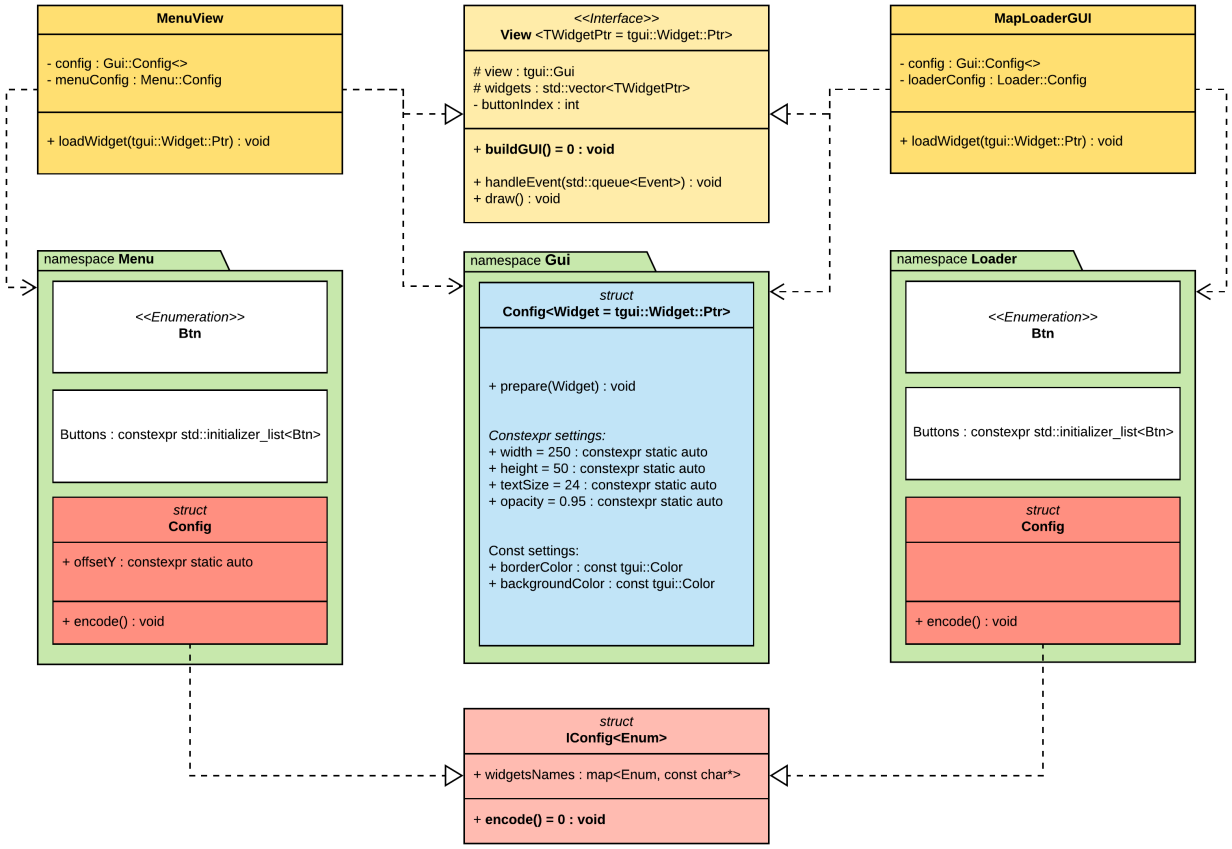
However, I've made some diagrams myself as a simplification over doxygen's:



5.14 States

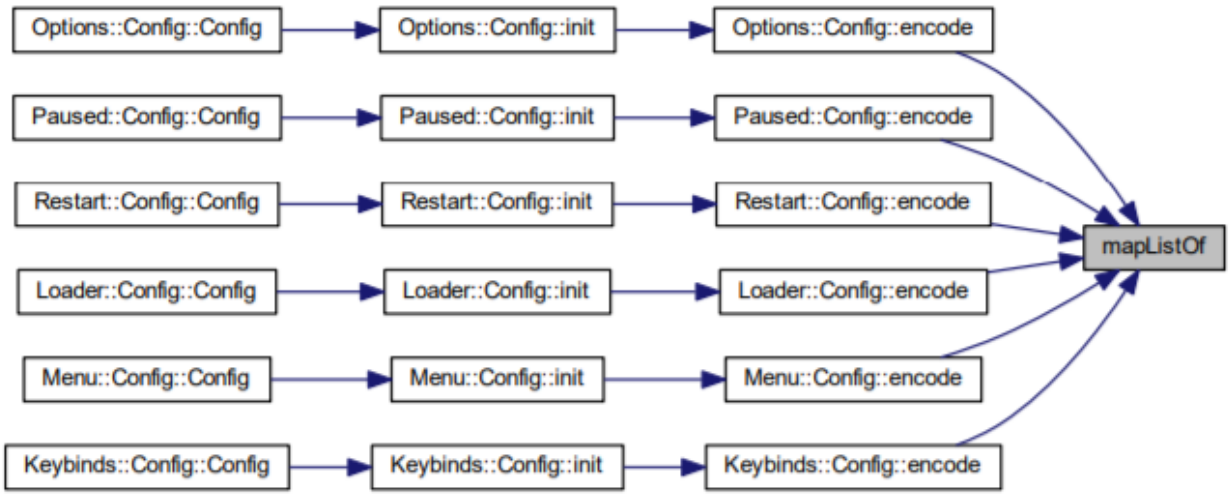

StateMachine
+ processInput() : void
+ update(float) : void
+ draw(Window&) : void
+ insert(shared_ptr<State>&) : int
+ erase(int) : void
+ switchTo(int) : void/bool
+ switchToDefault() : void/bool
- states : unordered_map<int, shared_ptr<State>>
- currentState : shared_ptr<State>
- statesNumber : int
1
0..*
<<interface>>
State
+ onCreate() = 0 : void
+ onDestroy() = 0 : void
+ onActivate() : void
+ onDeactivate() : void
+ processInput() : void
+ update(float dt) : void
+ draw(Window& window) : void
StateAbout
StateGame
StateKeybinds
StateMapLoader
StateMenu
StateOptions
StatePaused
StateRestart
namespace state
+ gameID : int
+ menuID : int
+ optionsID : int
insert(shared_ptr<State>&) : int

5.15 Views

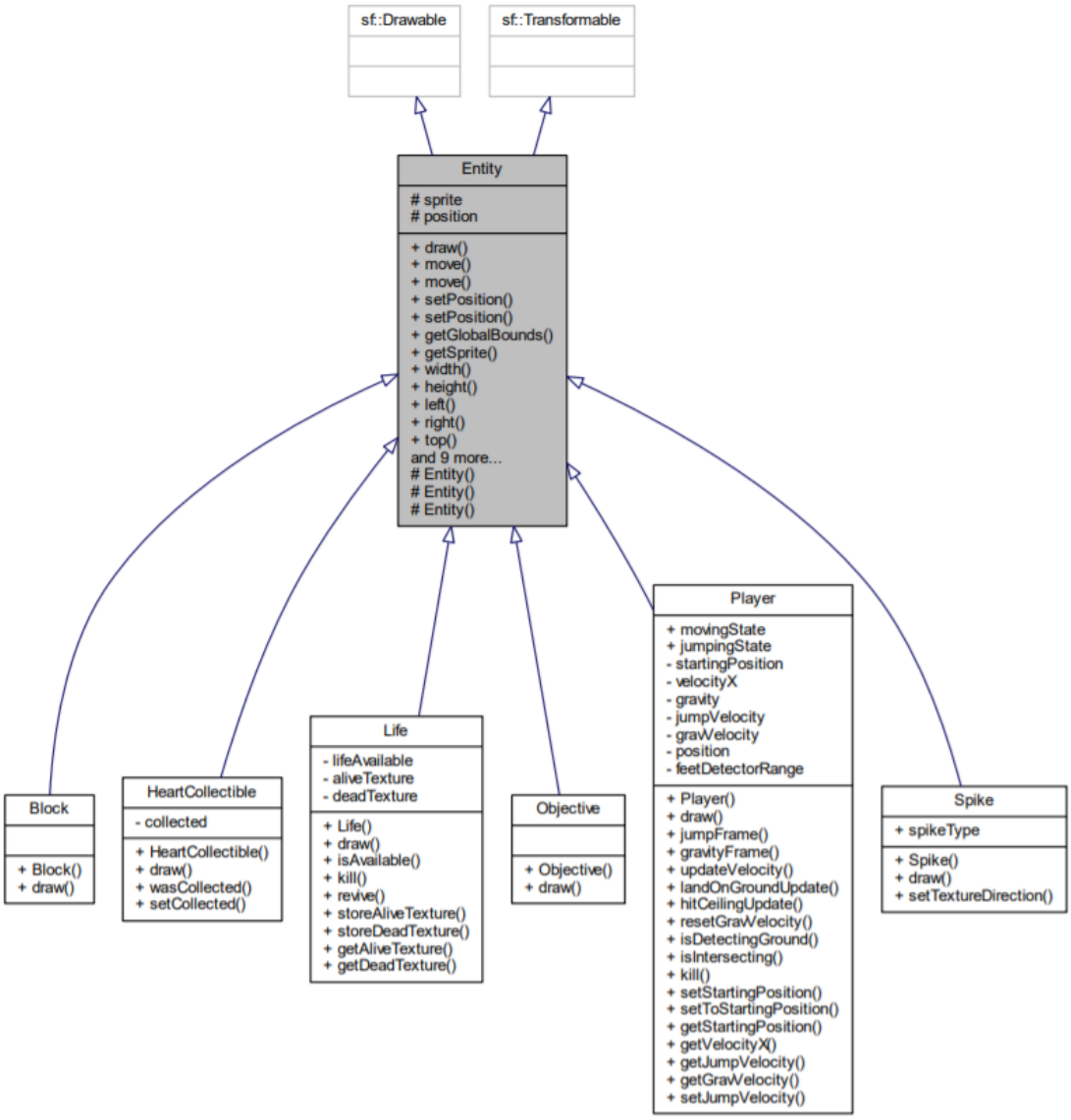

MenuView
- config : Gui::Config<>
- menuConfig : Menu::Config
+ loadWidget(tgui::Widget::Ptr) : void
<<Interface>>
View <TWidgetPtr = tgui::Widget::Ptr>
view : tgui::Gui
widgets : std::vector<TWidgetPtr>
- buttonIndex : int
+ buildGUI() = 0 : void
+ handleEvent(std::queue<Event>) : void
+ draw() : void
MapLoaderGUI
- config : Gui::Config<>
- loaderConfig : Loader::Config
+ loadWidget(tgui::Widget::Ptr) : void
namespace Menu
<<Enumeration>>
Btn
Buttons : constexpr std::initializer_list<Btn>
struct
Config
+ offsetY : constexpr static auto
+ encode() : void
namespace Gui
struct
Config<Widget = tgui::Widget::Ptr>
+ prepare(Widget) : void
Constexpr settings:
+ width = 250 : constexpr static auto
+ height = 50 : constexpr static auto
+ textSize = 24 : constexpr static auto
+ opacity = 0.95 : constexpr static auto
Const settings:
+ borderColor : const tgui::Color
+ backgroundColor : const tgui::Color
namespace Loader
<<Enumeration>>
Btn
Buttons : constexpr std::initializer_list<Btn>
struct
Config
+ encode() : void
struct
IConfig<Enum>
+ widgetsNames : map<Enum, const char*>
+ encode() = 0 : void

5.16 Utility

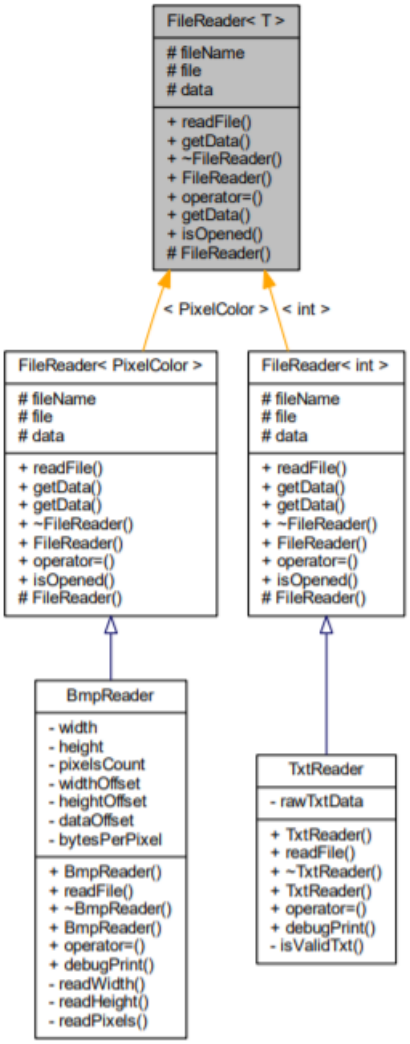
mapListOf caller diagram



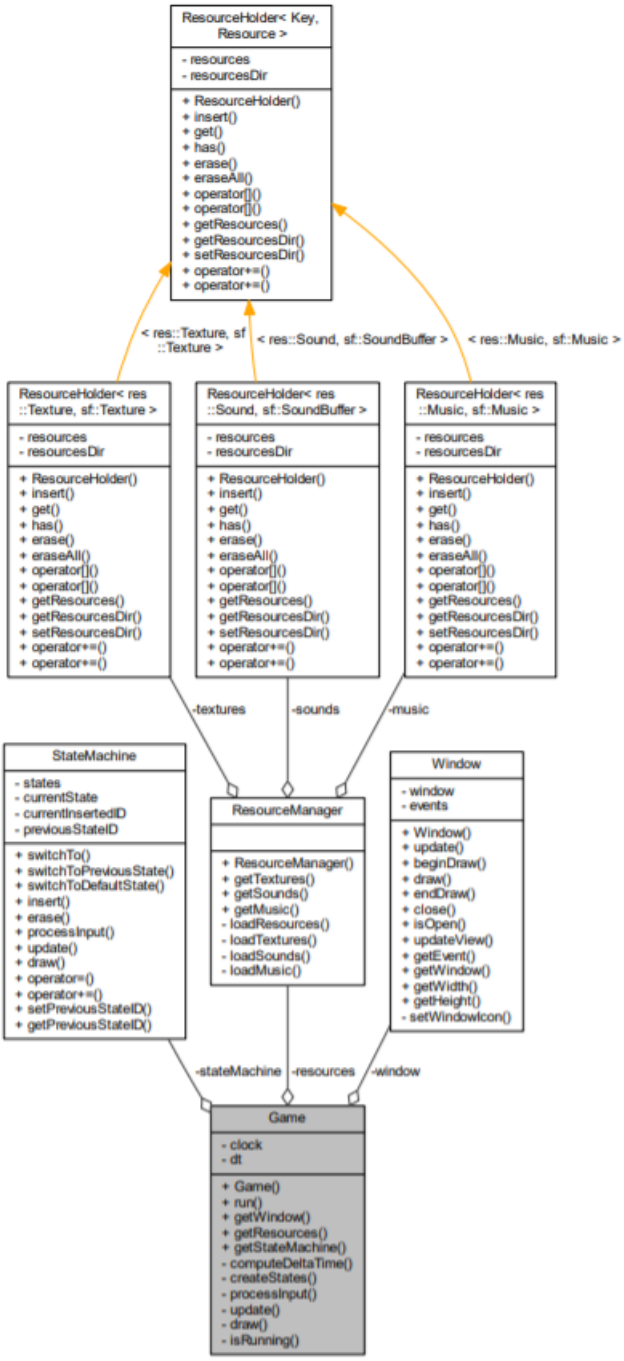
5.17 Entities



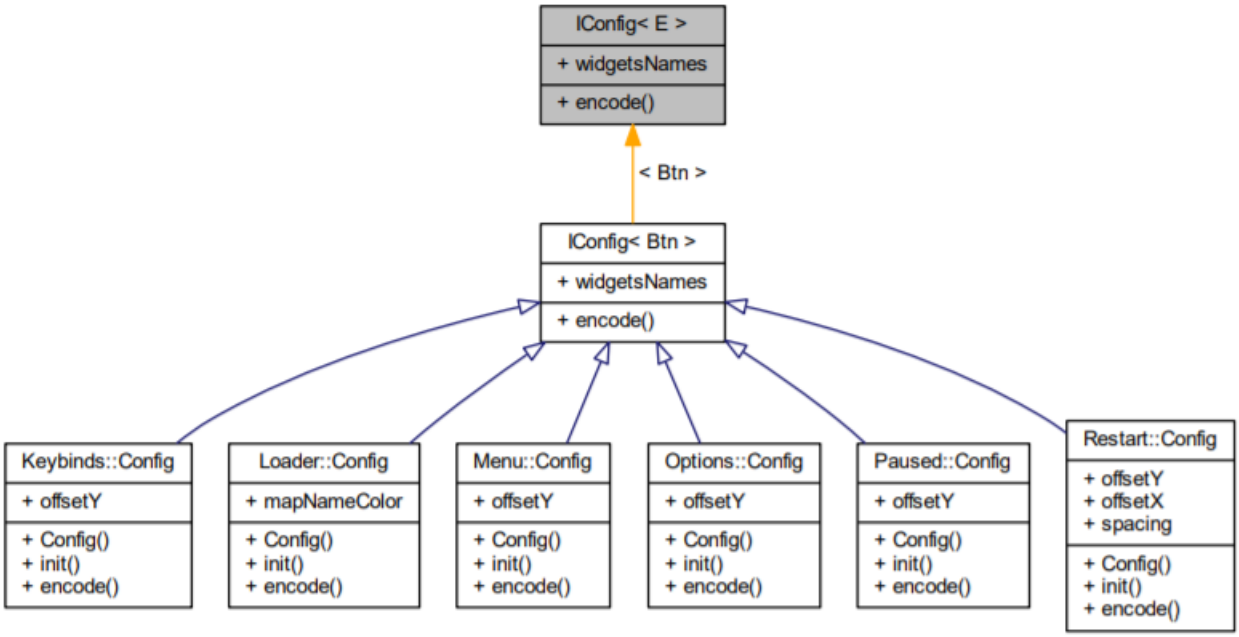
5.18 FileReader



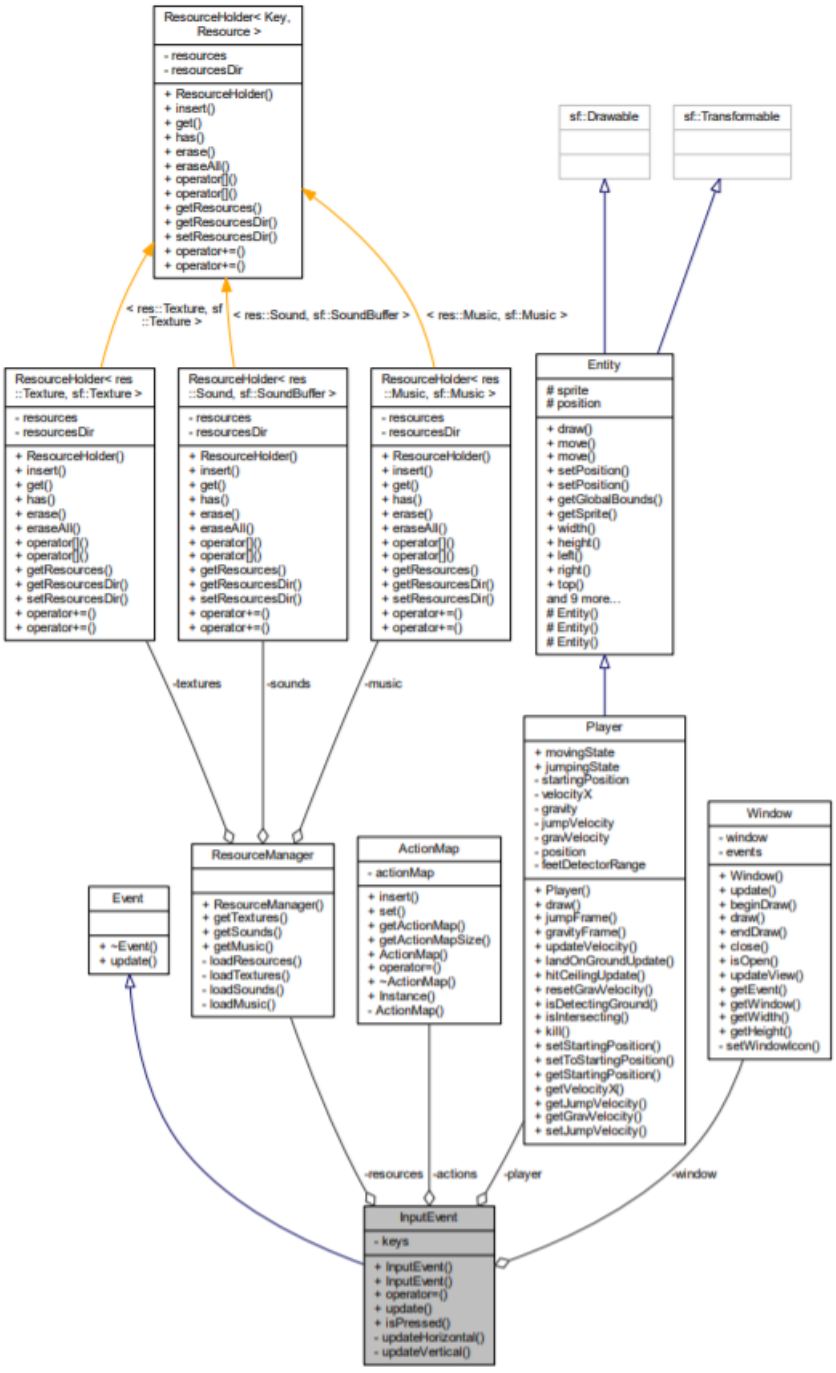
5.19 Game


ResourceHolder< Key, Resource >
- resources
- resourcesDir
+ ResourceHolder()
+ insert()
+ get()
+ has()
+ erase()
+ eraseAll()
+ operator[]()
+ operator[]()
+ getResources()
+ getResourcesDir()
+ setResourcesDir()
+ operator+=()
+ operator+=()
< res::Texture, sf ::Texture >
< res::Sound, sf::SoundBuffer >
< res::Music, sf::Music >
ResourceHolder< res ::Texture, sf::Texture >
ResourceHolder< res ::Sound, sf::SoundBuffer >
ResourceHolder< res ::Music, sf::Music >
-textures
-sounds
-music
StateMachine
- states
- currentState
- currentInsertedID
- previousStateID
+ switchTo()
+ switchToPreviousState()
+ switchToDefaultState()
+ insert()
+ erase()
+ processInput()
+ update()
+ draw()
+ operator=()
+ operator+=()
+ setPreviousStateID()
+ getPreviousStateID()
ResourceManager
+ ResourceManager()
+ getTextures()
+ getSounds()
+ getMusic()
- loadResources()
- loadTextures()
- loadSounds()
- loadMusic()
Window
- window
- events
+ Window()
+ update()
+ beginDraw()
+ draw()
+ endDraw()
+ close()
+ isOpen()
+ updateView()
+ getEvent()
+ getWindow()
+ getWidth()
+ getHeight()
- setWindowIcon()
-stateMachine
-resources
-window
Game
- clock
- dt
+ Game()
+ run()
+ getWindow()
+ getResources()
+ getStateMachine()
- computeDeltaTime()
- createStates()
- processInput()
- update()
- draw()
- isRunning()

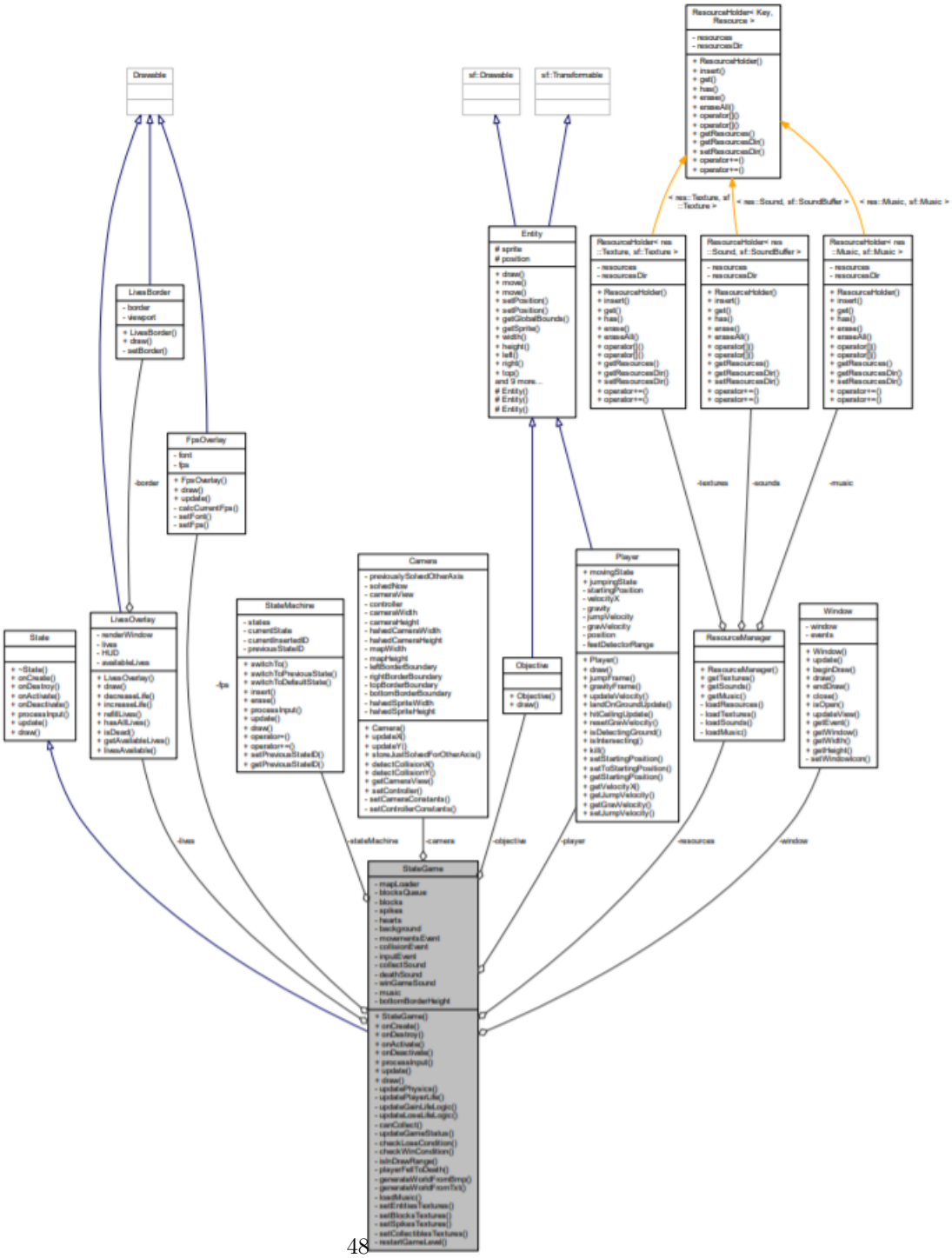
5.20 IConfg


IConfig< E >
+ widgetsNames
+ encode()
< Btn >
IConfig< Btn >
+ widgetsNames
+ encode()
Keybinds::Config
+ offsetY
+ Config()
+ init()
+ encode()
Loader::Config
+ mapNameColor
+ Config()
+ init()
+ encode()
Menu::Config
+ offsetY
+ Config()
+ init()
+ encode()
Options::Config
+ offsetY
+ Config()
+ init()
+ encode()
Paused::Config
+ offsetY
+ Config()
+ init()
+ encode()
Restart::Config
+ offsetY
+ offsetX
+ spacing
+ Config()
+ init()
+ encode()

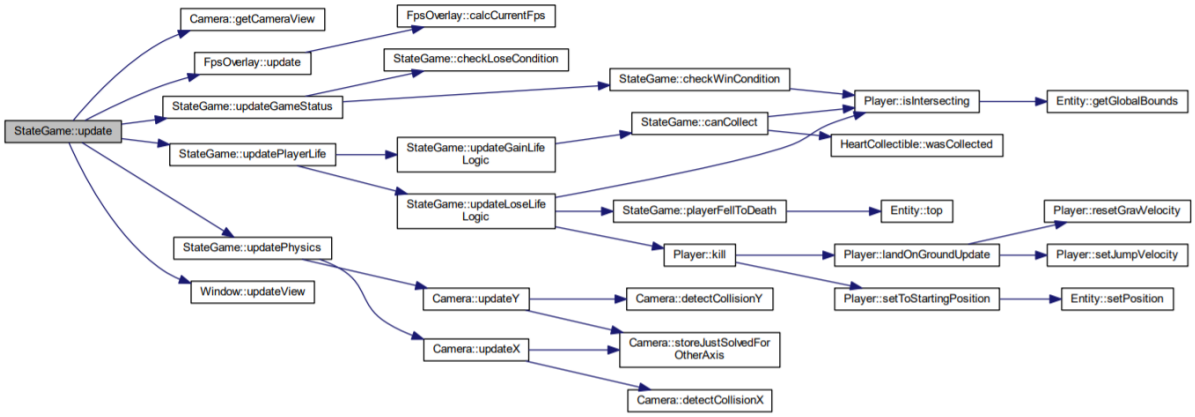
5.21 InputEvent



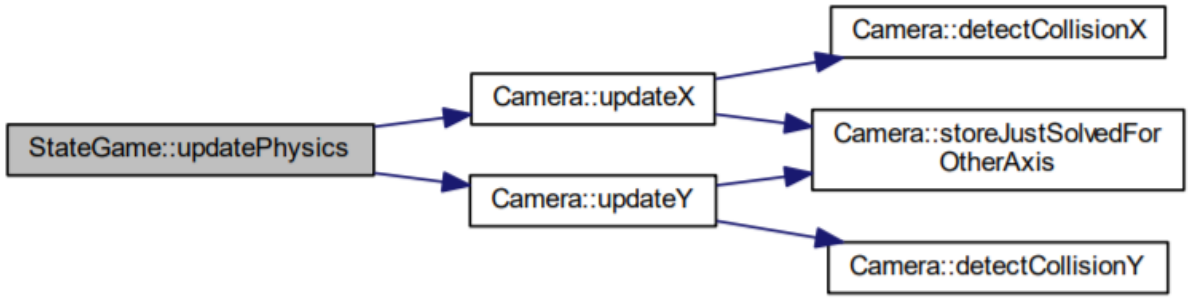
5.22 StateGame



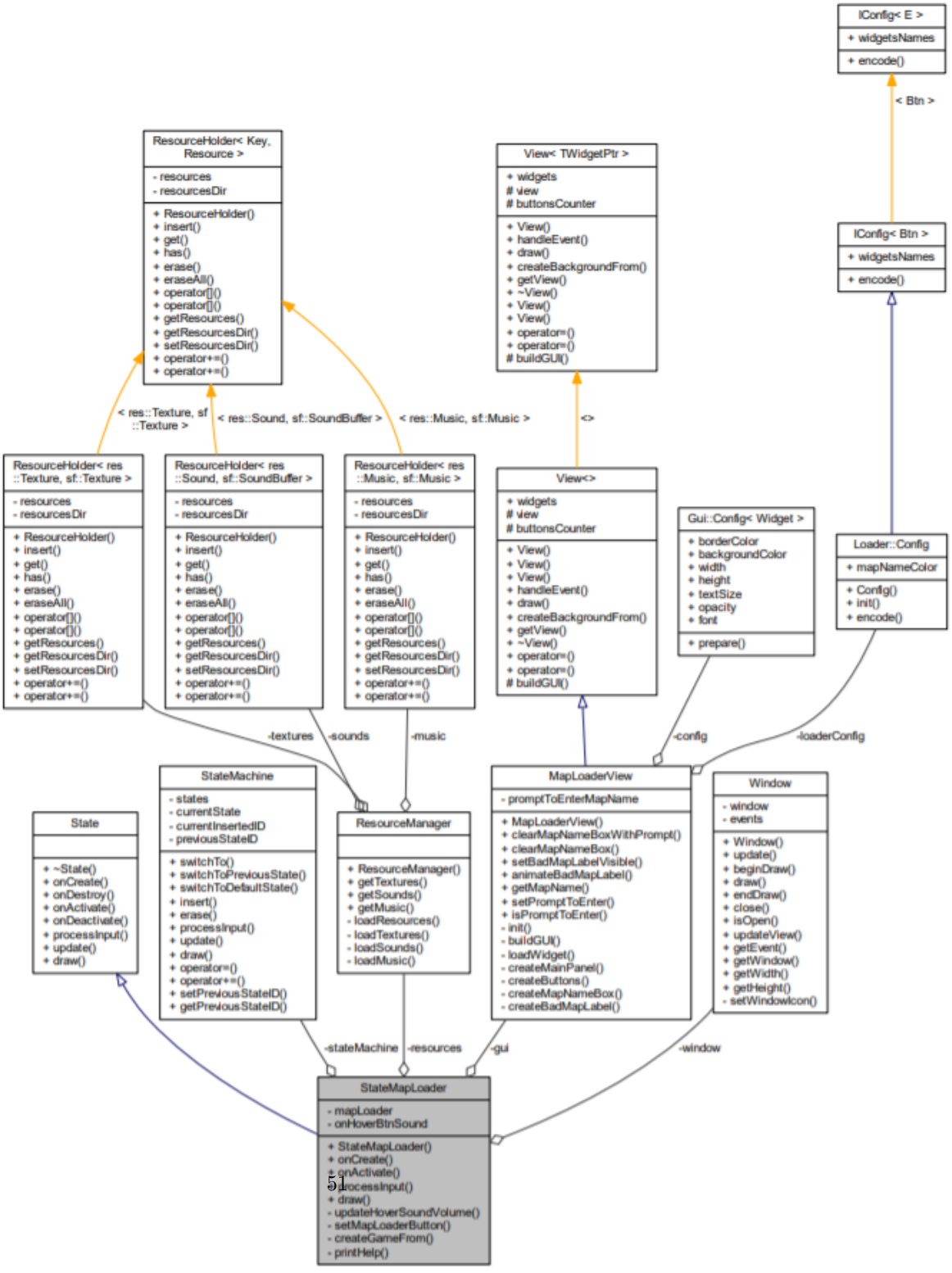
State Game update loop (main game loop):



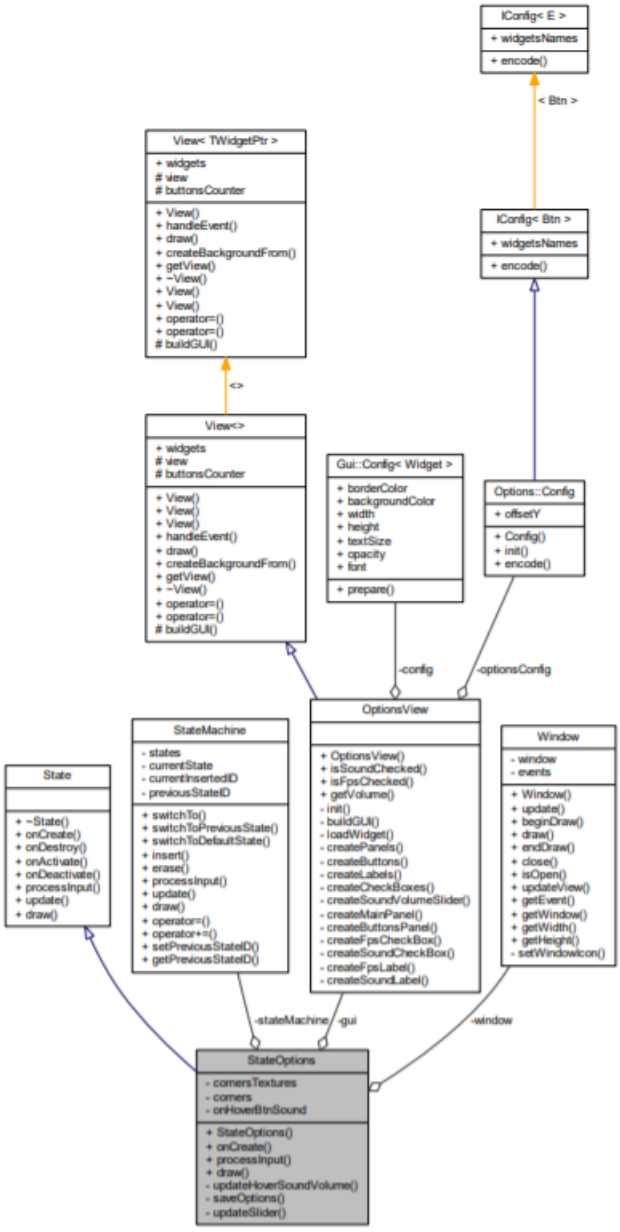
CameraUpdate:



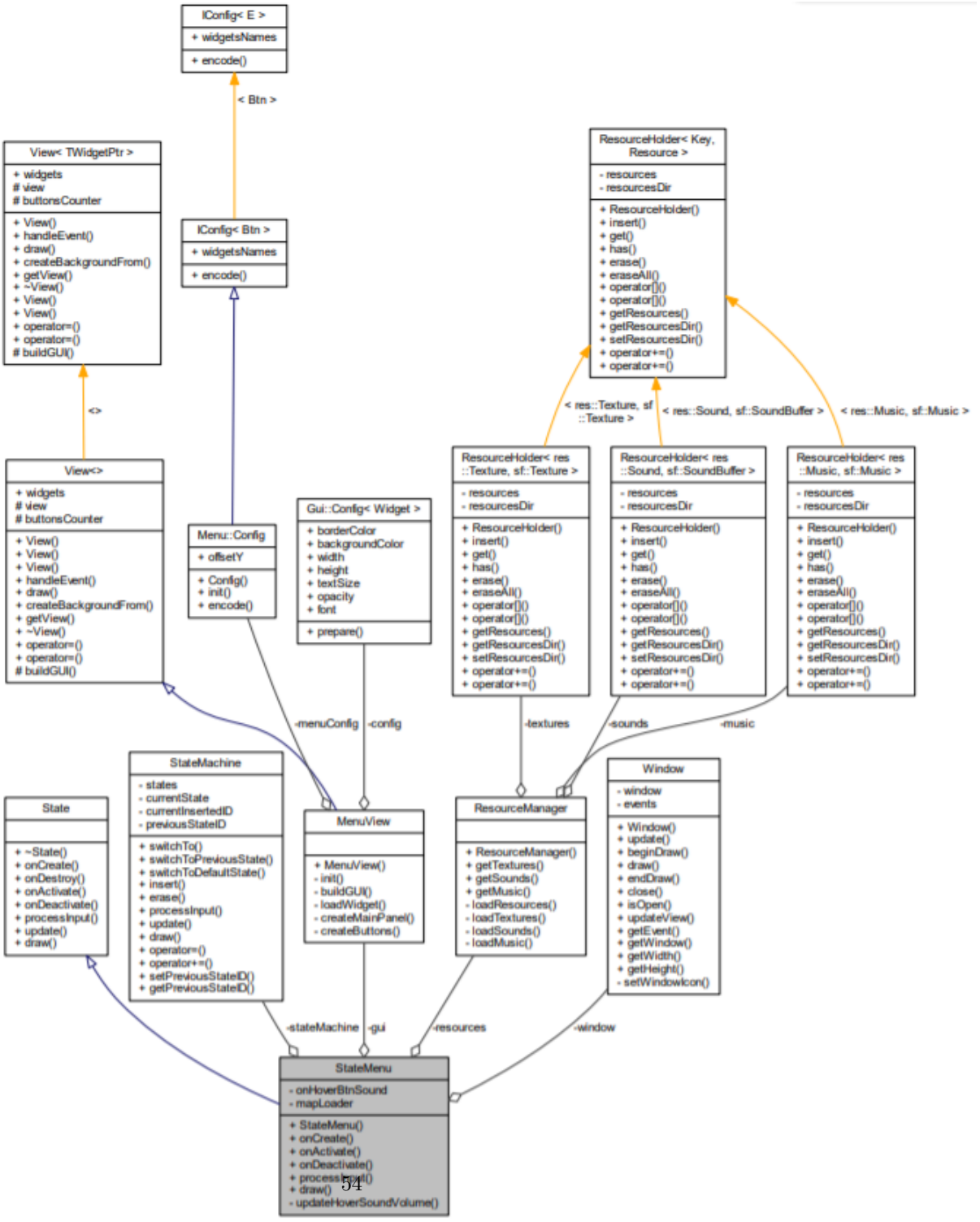
5.23 MapLoader



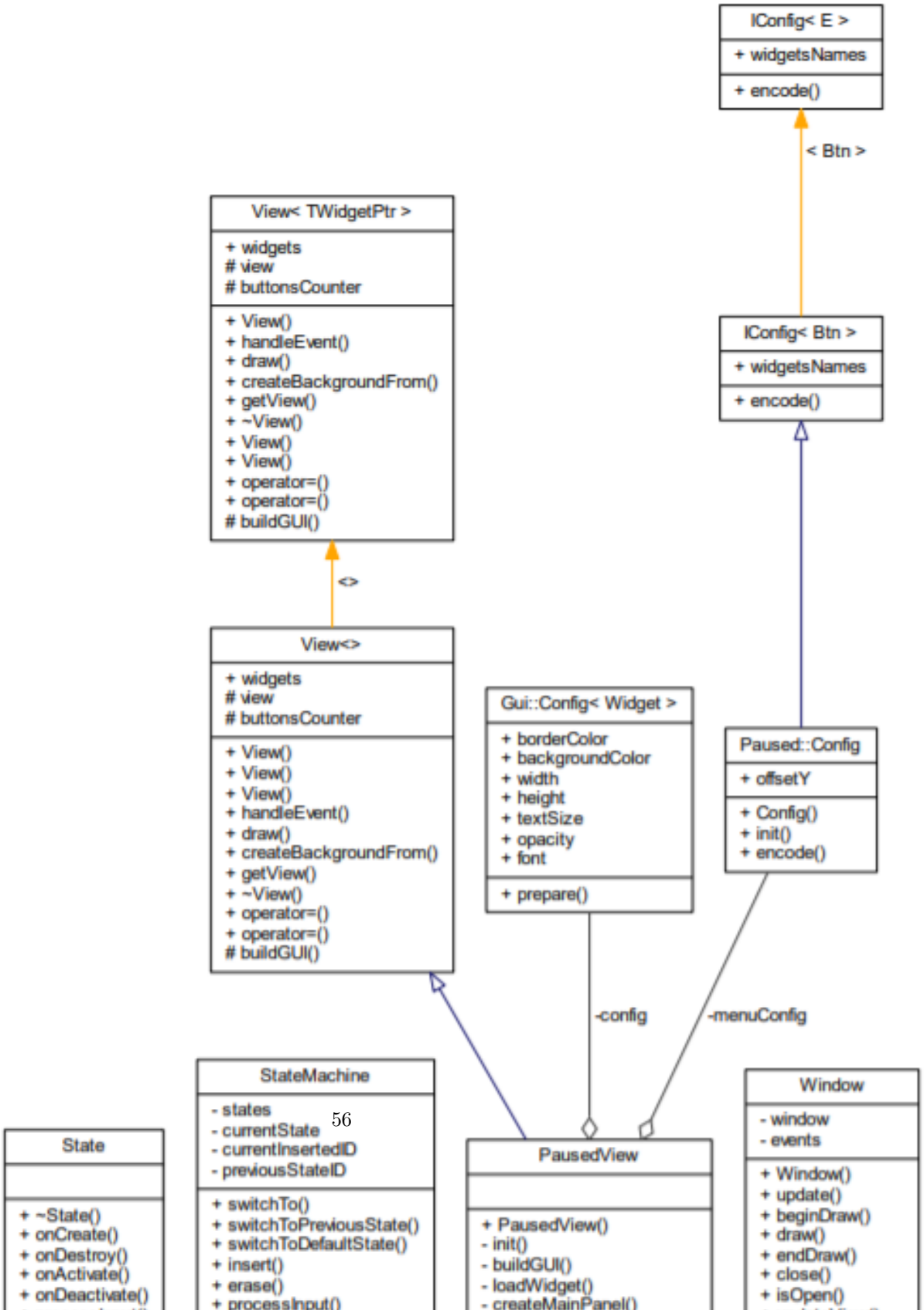
5.24 StateOptions



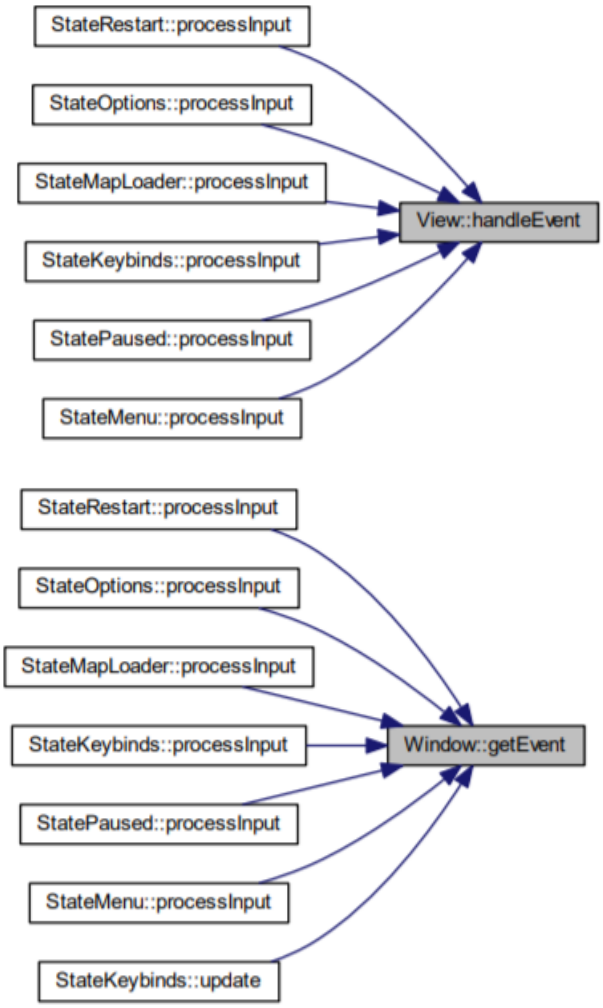
5.25 StateMenu

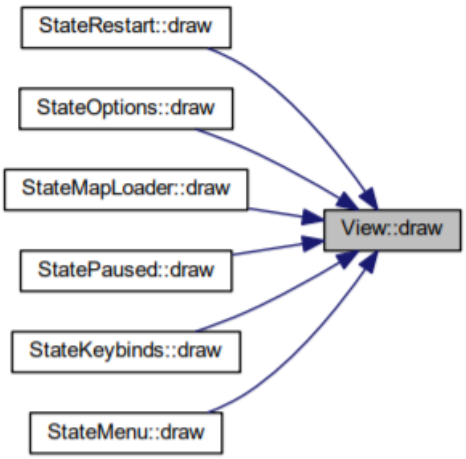


5.26 StatePaused

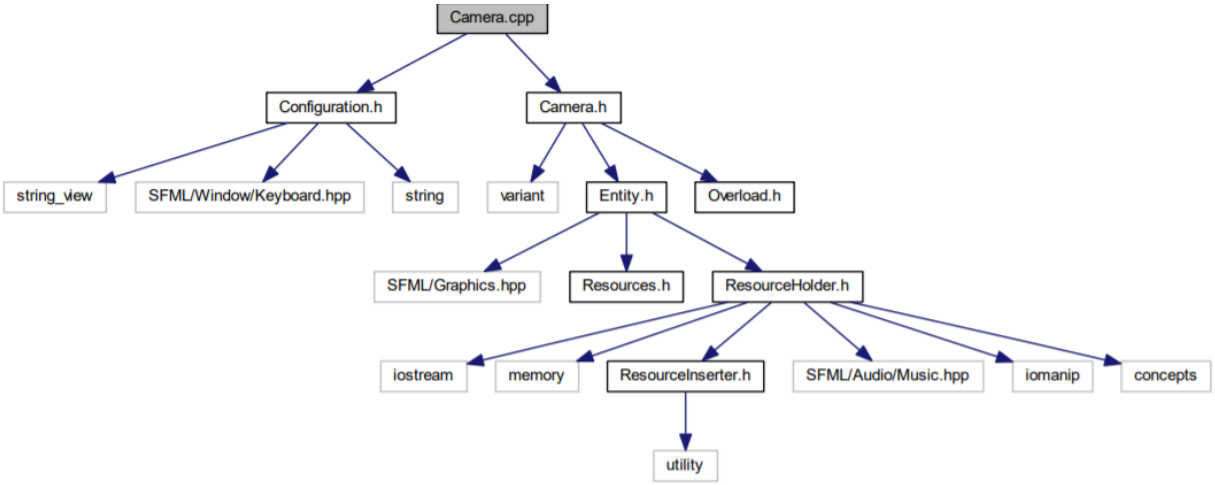


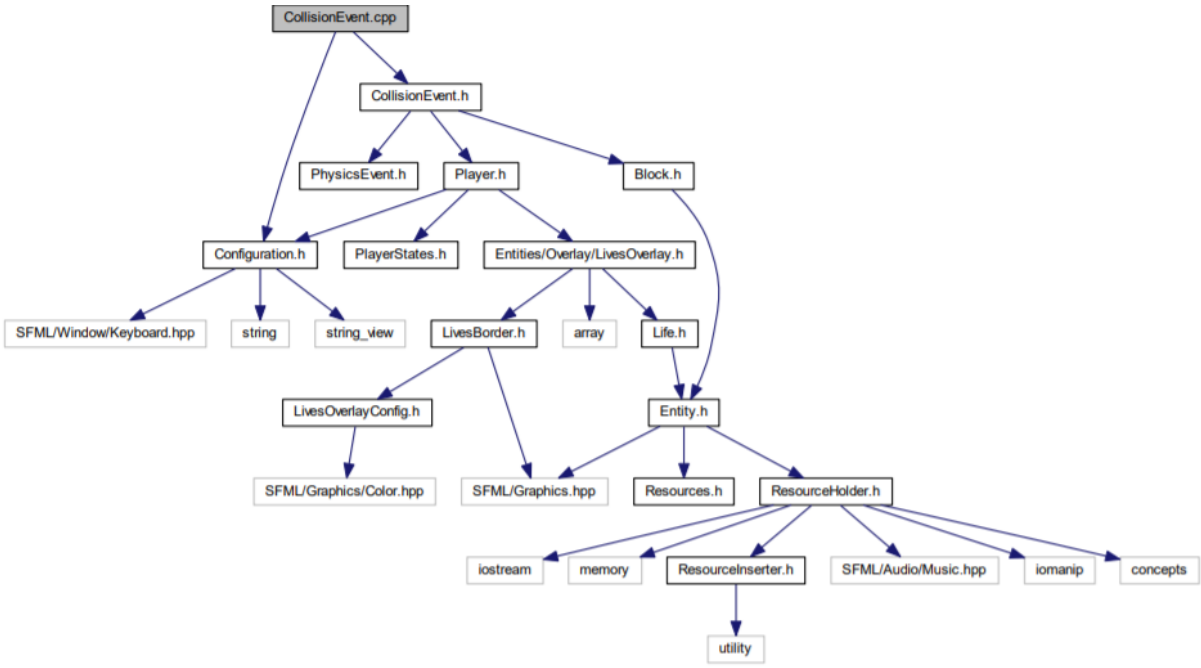
5.27 State Design Pattern

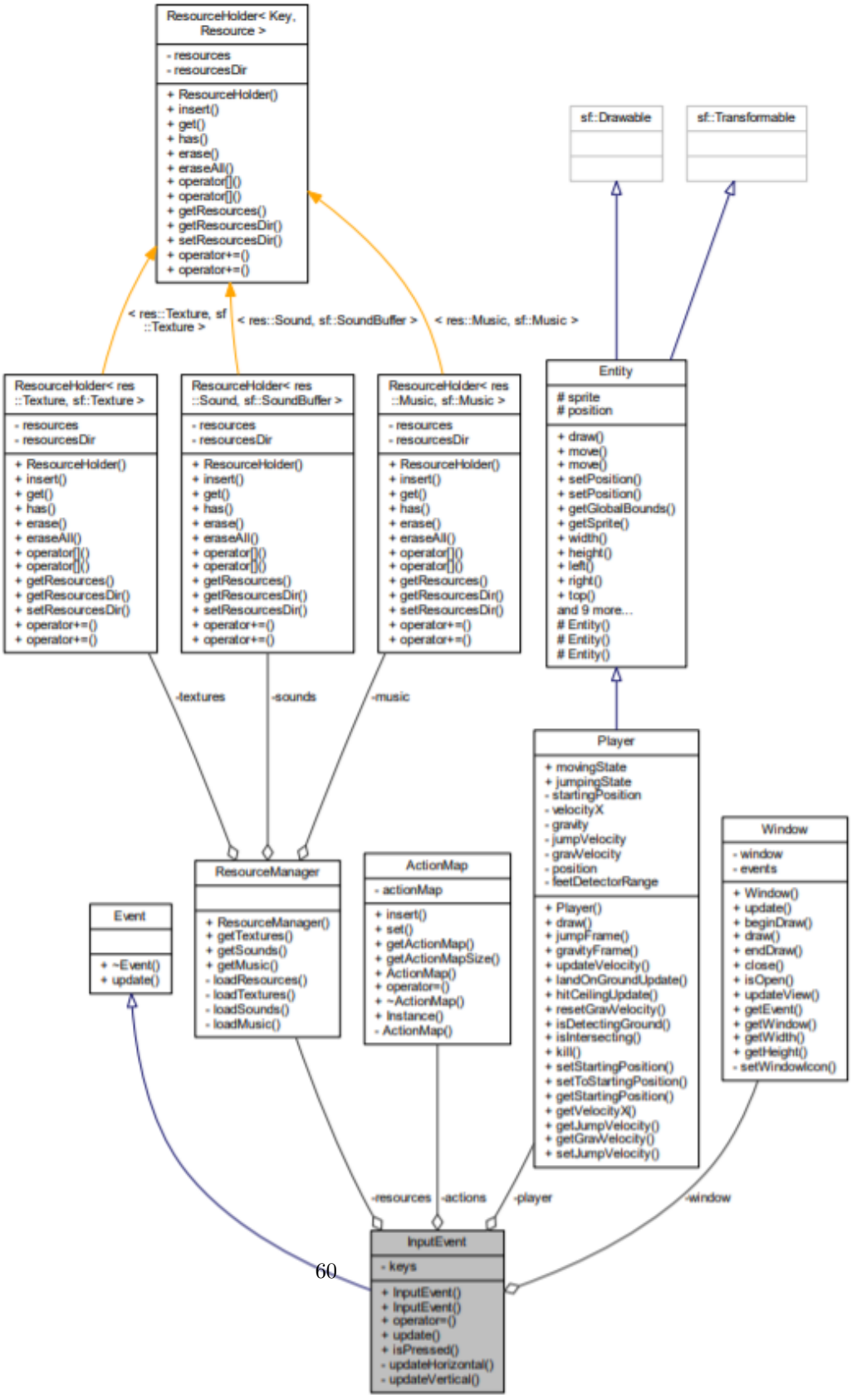




5.28 Include Dependencies




CollisionEvent.cpp
CollisionEvent.h
PhysicsEvent.h
Player.h
Block.h
Configuration.h
PlayerStates.h
Entities/Overlay/LivesOverlay.h
SFML/Window/Keyboard.hpp
string
string_view
LivesBorder.h
array
Life.h
LivesOverlayConfig.h
Entity.h
SFML/Graphics/Color.hpp
SFML/Graphics.hpp
Resources.h
ResourceHolder.h
iostream
memory
ResourceInserter.h
SFML/Audio/Music.hpp
iomanip
concepts
utility


ResourceHolder< Key, Resource >
- resources
- resourcesDir
+ ResourceHolder()
+ insert()
+ get()
+ has()
+ erase()
+ eraseAll()
+ operator[]()
+ operator[]()
+ getResources()
+ getResourcesDir()
+ setResourcesDir()
+ operator+=()
+ operator+=()
sf::Drawable
sf::Transformable
< res::Texture, sf ::Texture >
< res::Sound, sf::SoundBuffer >
< res::Music, sf::Music >
ResourceHolder< res ::Texture, sf::Texture >
ResourceHolder< res ::Sound, sf::SoundBuffer >
ResourceHolder< res ::Music, sf::Music >
Entity
sprite
position
+ draw()
+ move()
+ move()
+ setPosition()
+ setPosition()
+ getGlobalBounds()
+ getSprite()
+ width()
+ height()
+ left()
+ right()
+ top()
and 9 more...
Entity()
Entity()
Entity()
-textures
-sounds
-music
Player
+ movingState
+ jumpingState
- startingPosition
- velocityX
- gravity
- jumpVelocity
- gravVelocity
- position
- feetDetectorRange
+ Player()
+ draw()
+ jumpFrame()
+ gravityFrame()
+ updateVelocity()
+ landOnGroundUpdate()
+ hitCeilingUpdate()
+ resetGravVelocity()
+ isDetectingGround()
+ isIntersecting()
+ kill()
+ setStartingPosition()
+ setToStartingPosition()
+ getStartingPosition()
+ getVelocityX()
+ getJumpVelocity()
+ getGravVelocity()
+ setJumpVelocity()
Window
- window
- events
+ Window()
+ update()
+ beginDraw()
+ draw()
+ endDraw()
+ close()
+ isOpen()
+ updateView()
+ getEvent()
+ getWindow()
+ getWidth()
+ getHeight()
- setWindowIcon()
Event
+ ~Event()
+ update()
ResourceManager
+ ResourceManager()
+ getTextures()
+ getSounds()
+ getMusic()
- loadResources()
- loadTextures()
- loadSounds()
- loadMusic()
ActionMap
- actionMap
+ insert()
+ set()
+ getActionMap()
+ getActionMapSize()
+ ActionMap()
+ operator=()
+ ~ActionMap()
+ Instance()
- ActionMap()
-resources
-actions
-player
-window
InputEvent
- keys
+ InputEvent()
+ InputEvent()
+ operator=()
+ update()
+ isPressed()
- updateHorizontal()
- updateVertical()

6 Debugging

I have compiled the program on gcc 10.1.0 with all possible warning enabled, i.e.:
`-Wextra -pedantic -Wcast-align -Wcast-qual -Wctor-dtor-privacy [ . . . ]`

The program compiles with no warnings, unless C++17 is specified. In such a case, the following warning are messaged:

```
warning: range-based 'for' loops with initializer only available with '-std=c++2a'
or '-std=gnu++2a'
```

The program was somehow lacking proper testing, which is something to be improved.

The most difficult part was debuginning why sf::Music do not work (program was crashing at unpredictable times during launch), as I lost several hours to this problem.

The program does not leak memory as it never dynamically allocates with 'new' operator. Only smart pointers and RAII concept is being used. There are no cycling references within smart pointers.