CS202 The crossing road - Group 9

Generated by Doxygen 1.10.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 GUI Namespace Reference

4.2 ObstacleDataTables Namespace Reference

Variables

- const std::vector< ObstacleData > data = initializeObstacleData()

4.2.1 Variable Documentation

4.2.1.1 data

```
const std::vector<ObstacleData> ObstacleDataTables::data = initializeObstacleData()
```

Definition at line 9 of file ObstacleManagement.cpp.

Chapter 5

Class Documentation

5.1 CharacterMover Struct Reference

Collaboration diagram for CharacterMover:

Public Member Functions

- CharacterMover (float vx, float vy)
- void operator() (Character &character, sf::Time) const

Public Attributes

- sf::Vector2f velocity

5.1.1 Detailed Description

Definition at line 11 of file Player.cpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 CharacterMover()

```
CharacterMover::CharacterMover (
            float vx,
            float vy )  [inline]
```

Definition at line 12 of file Player.cpp.

5.1.3 Member Function Documentation

5.1.3.1 operator()()

```
void CharacterMover::operator() (
            Character & character,
            sf::Time  ) const  [inline]
```

Definition at line 14 of file Player.cpp.

5.1.4 Member Data Documentation

5.1.4.1 velocity

```
sf::Vector2f CharacterMover::velocity
```

Definition at line 18 of file Player.cpp.

The documentation for this struct was generated from the following file:

- Source/Player.cpp

Chapter 6

File Documentation

6.1 Source/Animation.cpp File Reference

```
#include <Animation.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/Texture.hpp>
```

Include dependency graph for Animation.cpp:

6.2 Animation.cpp

Go to the documentation of this file.

```
00001 #include <Animation.hpp>
00002
00003 #include <SFML/Graphics/RenderTarget.hpp>
00004 #include <SFML/Graphics/Texture.hpp>
00005
00006
00007 Animation::Animation()
00008 : mSprite()
00009 , mFrameSize()
00010 , mNumFrames(0)
00011 , mCurrentFrame(0)
00012 , mDuration(sf::Time::Zero)
00013 , mElapsedTime(sf::Time::Zero)
00014 , mRepeat(false)
00015 {
00016 }
00017
00018 Animation::Animation(const sf::Texture& texture)
00019 : mSprite(texture)
00020 , mFrameSize()
00021 , mNumFrames(0)
00022 , mCurrentFrame(0)
00023 , mDuration(sf::Time::Zero)
00024 , mElapsedTime(sf::Time::Zero)
00025 , mRepeat(false)
00026 , isBuiltYet(true)
00027 {
00028 }
00029
00030 bool Animation::isBuilt() {
00031     return isBuiltYet;
00032 }
00033
00034 void Animation::setTexture(const sf::Texture& texture)
00035 {
00036     isBuiltYet=true;
00037     mSprite.setTexture(texture);
00038 }
00039
00040 const sf::Texture* Animation::getTexture() const
```

```
00041 {
00042     return mSprite.getTexture();
00043 }
00044 void Animation::setAnimation(const std::string& filename, int numFrame, int x, int y){
00045     sf::Texture* texture = new sf::Texture();
00046     if (!texture->loadFromFile(filename)){
00047         throw "Invalid " + filename;
00048     }
00049     setTexture(*texture);
00050     setNumFrames(numFrame);
00051     setFrameSize(sf::Vector2i(x, y));
00052     setRepeating(true);
00053     setDuration(sf::seconds(1));
00054 }
00055 void Animation::setFrameSize(sf::Vector2i frameSize)
00056 {
00057     mFrameSize = frameSize;
00058 }
00059
00060 sf::Vector2i Animation::getFrameSize() const
00061 {
00062     return mFrameSize;
00063 }
00064
00065 void Animation::setNumFrames(std::size_t numFrames)
00066 {
00067     mNumFrames = numFrames;
00068 }
00069
00070 std::size_t Animation::getNumFrames() const
00071 {
00072     return mNumFrames;
00073 }
00074
00075 void Animation::setDuration(sf::Time duration)
00076 {
00077     mDuration = duration;
00078 }
00079
00080 sf::Time Animation::getDuration() const
00081 {
00082     return mDuration;
00083 }
00084
00085 void Animation::setRepeating(bool flag)
00086 {
00087     mRepeat = flag;
00088 }
00089
00090 bool Animation::isRepeating() const
00091 {
00092     return mRepeat;
00093 }
00094
00095 void Animation::restart()
00096 {
00097     isFinishedFlag = false;
00098     mCurrentFrame = 0;
00099 }
00100
00101
00102 bool Animation::isFinished() const
00103 {
00104     return isFinishedFlag;
00105 }
00106
00107 sf::FloatRect Animation::getLocalBounds() const
00108 {
00109     float width = static_cast<float>(std::abs(getFrameSize().x));
00110     float height = static_cast<float>(std::abs(getFrameSize().y));
00111     return sf::FloatRect(0.f, 0.f, width, height);
00112     // return sf::FloatRect(getOrigin(), static_cast<sf::Vector2f>(getFrameSize()));
00113 }
00114
00115 sf::FloatRect Animation::getGlobalBounds() const
00116 {
00117     return getTransform().transformRect(getLocalBounds());
00118 }
00119
00120 void Animation::update(sf::Time dt)
00121 {
00122     sf::Time timePerFrame = mDuration / static_cast<float>(mNumFrames);
00123     mElapsedTime += dt;
00124
00125     sf::Vector2i textureBounds(mSprite.getTexture()->getSize());
00126     sf::IntRect textureRect = mSprite.getTextureRect();
00127
```

```
00128     if (mCurrentFrame == 0)
00129         textureRect = sf::IntRect(0, 0, mFrameSize.x, mFrameSize.y);
00130 
00131     while (mElapsedTime >= timePerFrame && (mCurrentFrame <= mNumFrames || mRepeat))
00132     {
00133         textureRect.left += textureRect.width;
00134 
00135         if (textureRect.left + textureRect.width > textureBounds.x)
00136         {
00137             textureRect.left = 0;
00138             textureRect.top += textureRect.height;
00139         }
00140 
00141         int tmp = (textureBounds.x + textureRect.width - 1) / textureRect.width;
00142         textureRect.left = textureRect.width * (mCurrentFrame%tmp);
00143         textureRect.top = textureRect.height * (mCurrentFrame/tmp);
00144 
00145         mElapsedTime -= timePerFrame;
00146         if (mRepeat)
00147         {
00148             if (mCurrentFrame>=mNumFrames) {
00149                 isFinishedFlag = true;
00150             }
00151 
00152             mCurrentFrame = (mCurrentFrame + 1) % mNumFrames;
00153 
00154             if (mCurrentFrame == 0)
00155                 textureRect = sf::IntRect(0, 0, mFrameSize.x, mFrameSize.y);
00156         }
00157         else
00158         {
00159             mCurrentFrame++;
00160             if (mCurrentFrame>=mNumFrames) {
00161                 mCurrentFrame=mNumFrames-1;
00162                 isFinishedFlag = true;
00163             }
00164             if (mNumFrames<0) {
00165                 mNumFrames=0;
00166             }
00167         }
00168     }
00169 
00170 
00171     mSprite.setTextureRect(textureRect);
00172 
00173 }
00174 
00175 void Animation::draw(sf::RenderTarget& target, sf::RenderStates states) const
00176 {
00177     if (isHide) return;
00178     states.transform *= getTransform();
00179     target.draw(mSprite, states);
00180 }
00181 
00182 void Animation::hide() {
00183     isHide=true;
00184 }
00185 
00186 void Animation::show() {
00187     isHide=false;
00188 }
00189 
00190 bool Animation::isShow() {
00191     return !isHide;
00192 }
```

6.3 Source/Application.cpp File Reference

```
#include <Application.hpp>
#include <Utility.hpp>
#include <State.hpp>
#include <StateIdentifiers.hpp>
#include <TitleState.hpp>
#include <CharacterState.hpp>
#include <MapState.hpp>
#include <GameState.hpp>
#include <MenuState.hpp>
```

```
#include <SettingState.hpp>
#include <HighScoreState.hpp>
#include <CreditState.hpp>
#include <PauseState.hpp>
#include <Const.hpp>
#include <GameOverState.hpp>
#include <CountDownState.hpp>
#include <SavingState.hpp>
#include <GameLevel.hpp>
```

Include dependency graph for Application.cpp:

6.4 Application.cpp

Go to the documentation of this file.

```
00001 #include <Application.hpp>
00002 #include <Utility.hpp>
00003 #include <State.hpp>
00004 #include <StateIdentifiers.hpp>
00005 #include <TitleState.hpp>
00006 #include <CharacterState.hpp>
00007 #include <MapState.hpp>
00008 #include <GameState.hpp>
00009 #include <MenuState.hpp>
00010 #include <SettingState.hpp>
00011 #include <HighScoreState.hpp>
00012 #include <CreditState.hpp>
00013 #include <PauseState.hpp>
00014 #include <Const.hpp>
00015
00016
00017 #include <GameOverState.hpp>
00018 #include <CountDownState.hpp>
00019 #include <SavingState.hpp>
00020
00021 #include <GameLevel.hpp>
00022 const sf::Time Application::TimePerFrame = sf::seconds(1.f/60.f);
00023
00024 Application::Application() : mWindow(sf::VideoMode(Constants::WindowWidth, Constants::WindowHeight),
      "States", sf::Style::Close), mTextures(), mFonts(), mPlayer(), mMusic(), mSounds(),
      mStateStack(State::Context(mWindow, mTextures, mFonts, mPlayer, mMusic, mSounds)), mStatisticsText(),
      mStatisticsUpdateTime(), mStatisticsNumFrames(0) {
00025     mWindow.setKeyRepeatEnabled(false);
00026
00027     mFonts.load(Fonts::Main, "Media/Fonts/Sansation.ttf");
00028
00029     mTextures.load(Textures::TitleScreen, "Media/Textures/TitleScreen.png");
00030     mTextures.load(Textures::Background, "Media/Textures/Background.png");
00031     mTextures.load(Textures::Title, "Media/Textures/Title.png");
00032     mTextures.load(Textures::Cloud1, "Media/Textures/Cloud1.png");
00033     mTextures.load(Textures::Cloud2, "Media/Textures/Cloud2.png");
00034     mTextures.load(Textures::Cloud3, "Media/Textures/Cloud3.png");
00035     mTextures.load(Textures::Cat, "Media/Textures/Cat.png");
00036     mTextures.load(Textures::Button, "Media/Textures/Button.png");
00037     mTextures.load(Textures::ButtonTouch, "Media/Textures/ButtonTouch.png");
00038     mTextures.load(Textures::ButtonPressed, "Media/Textures/ButtonPressed.png");
00039     mTextures.load(Textures::Key1, "Media/Textures/Key1.png");
00040     mTextures.load(Textures::Key2, "Media/Textures/Key2.png");
00041     mTextures.load(Textures::HighScore, "Media/Textures/HighScore.png");
00042     mTextures.load(Textures::Character, "Media/Textures/Character.png");
00043     mTextures.load(Textures::Countdown, "Media/Textures/Countdown.png");
00044     mTextures.load(Textures::Map, "Media/Textures/Map.png");
00045     mTextures.load(Textures::Credit,"Media/Textures/Credit.png");
00046     mTextures.load(Textures::Sound1, "Media/Textures/Sound1.png");
00047     mTextures.load(Textures::Sound2, "Media/Textures/Sound2.png");
00048
00049     mTextures.load(Textures::Winter, "Media/Textures/Winter.png");
00050     mTextures.load(Textures::Autumn, "Media/Textures/Autumn.png");
00051     mTextures.load(Textures::Spring, "Media/Textures/Spring.png");
00052     mTextures.load(Textures::Atlantis, "Media/Textures/Atlantis.png");
00053     mTextures.load(Textures::Jura, "Media/Textures/Jura.png");
00054
00055     mStatisticsText.setFont(mFonts.get(Fonts::Main));
00056     mStatisticsText.setPosition(5.f, 5.f);
00057     mStatisticsText.setCharacterSize(10u);
00058
00059     registerStates();
00060     mStateStack.pushState(States::Title);
```

```
00061
00062     mMusic.setVolume(25.f);
00063 }
00064
00065 void Application::run() {
00066     sf::Clock clock;
00067     sf::Time timeSinceLastUpdate = sf::Time::Zero;
00068
00069     while (mWindow.isOpen()) {
00070         sf::Time dt = clock.restart();
00071         timeSinceLastUpdate += dt;
00072         while (timeSinceLastUpdate > TimePerFrame) {
00073             timeSinceLastUpdate -= TimePerFrame;
00074
00075             processInput();
00076             update(TimePerFrame);
00077
00078             if (mStateStack.isEmpty())
00079                 mWindow.close();
00080         }
00081
00082         updateStatistics(dt);
00083         render();
00084     }
00085 }
00086
00087 void Application::processInput() {
00088     sf::Event event;
00089     while (mWindow.pollEvent(event)) {
00090         mStateStack.handleEvent(event);
00091
00092         if (event.type == sf::Event::Closed)
00093             mWindow.close();
00094     }
00095 }
00096 void Application::update(sf::Time dt) {
00097     mStateStack.update(dt);
00098 }
00099
00100 void Application::render() {
00101     mWindow.clear();
00102
00103     mStateStack.draw();
00104
00105     mWindow.setView(mWindow.getDefaultView());
00106     mWindow.draw(mStatisticsText);
00107
00108     // sf::Text text;
00109     // text.setFont(mFonts.get(Fonts::Main));
00110     // text.setPosition(10.f, 10.f);
00111     // text.setCharacterSize(20u);
00112     // text.setString("Level: " + std::to_string(gameLevel.getLevel()));
00113
00114     // mWindow.draw(text);
00115
00116     mWindow.display();
00117 }
00118
00119 void Application::updateStatistics(sf::Time dt) {
00120     mStatisticsUpdateTime += dt;
00121     mStatisticsNumFrames += 1;
00122
00123     if (mStatisticsUpdateTime >= sf::seconds(1.0f)) {
00124         mStatisticsText.setString("FPS: " + toString(mStatisticsNumFrames));
00125
00126         mStatisticsUpdateTime -= sf::seconds(1.0f);
00127         mStatisticsNumFrames = 0;
00128     }
00129 }
00130
00131 void Application::registerStates() {
00132     mStateStack.registerState<TitleState>(States::Title);
00133     mStateStack.registerState<MenuState>(States::Menu);
00134     mStateStack.registerState<CharacterState>(States::Character);
00135     mStateStack.registerState<MapState>(States::Map);
00136     mStateStack.registerState<SettingState>(States::Setting);
00137     mStateStack.registerState<CreditState>(States::Credit);
00138     mStateStack.registerState<HighScoreState>(States::Score);
00139     mStateStack.registerState<GameState>(States::Game);
00140     mStateStack.registerState<PauseState>(States::Pause);
00141     mStateStack.registerState<GameOverState>(States::GameOver);
00142     mStateStack.registerState<CountDownState>(States::CountDown);
00143     mStateStack.registerState<SavingState>(States::Saving);
00144 }
```

6.5 Source/Button.cpp File Reference

```
#include <Button.hpp>
#include <Utility.hpp>
#include <SoundPlayer.hpp>
#include <SFML/Window/Event.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for Button.cpp:

Namespaces

- namespace GUI

6.6 Button.cpp

Go to the documentation of this file.

```
00001 #include <Button.hpp>
00002 #include <Utility.hpp>
00003 #include <SoundPlayer.hpp>
00004
00005 #include <SFML/Window/Event.hpp>
00006 #include <SFML/Graphics/RenderStates.hpp>
00007 #include <SFML/Graphics/RenderTarget.hpp>
00008
00009
00010 namespace GUI
00011 {
00012
00013 Button::Button(State::Context context)
00014 : mCallback()
00015 , mNormalTexture(context.textures->get(Textures::Button))
00016 , mSelectedTexture(context.textures->get(Textures::ButtonTouch))
00017 , mPressedTexture(context.textures->get(Textures::ButtonPressed))
00018 , mSprite()
00019 , mText("", context.fonts->get(Fonts::Main), 16)
00020 , mIsToggle(false)
00021 , mSounds(*context.sounds)
00022 {
00023     mSprite.setTexture(mNormalTexture);
00024
00025     sf::FloatRect bounds = mSprite.getLocalBounds();
00026     mText.setPosition(bounds.width / 2.f, bounds.height / 2.f);
00027 }
00028
00029 void Button::setCallback(Callback callback)
00030 {
00031     mCallback = std::move(callback);
00032 }
00033
00034 void Button::setText(const std::string& text)
00035 {
00036     mText.setString(text);
00037     mText.setFillColor(sf::Color::Black);
00038     centerOrigin(mText);
00039 }
00040 void Button::setText(const std::string&text, unsigned int sizeText){
00041     mText.setString(text);
00042     mText.setFillColor(sf::Color::Black);
00043     mText.setCharacterSize(sizeText);
00044     centerOrigin(mText);
00045 }
00046 void Button::setToggle(bool flag)
00047 {
00048     mIsToggle = flag;
00049 }
00050
00051 bool Button::isSelectable() const
00052 {
00053     return true;
00054 }
00055
```

```
00056 void Button::select()
00057 {
00058     Component::select();
00059
00060     mSprite.setTexture(mSelectedTexture);
00061 }
00062
00063 void Button::deselect()
00064 {
00065     Component::deselect();
00066
00067     mSprite.setTexture(mNormalTexture);
00068 }
00069
00070 void Button::activate()
00071 {
00072     Component::activate();
00073
00074     // If we are toggle then we should show that the button is pressed and thus "toggled".
00075     if (mIsToggle)
00076         mSprite.setTexture(mPressedTexture);
00077
00078     if (mCallback)
00079         mCallback();
00080
00081     // If we are not a toggle then deactivate the button since we are just momentarily activated.
00082     if (!mIsToggle)
00083         deactivate();
00084
00085     mSounds.play(SoundEffects::Button);
00086
00087 }
00088
00089 void Button::deactivate()
00090 {
00091     Component::deactivate();
00092
00093     if (mIsToggle)
00094     {
00095         // Reset texture to right one depending on if we are selected or not.
00096         if (isSelected())
00097             mSprite.setTexture(mSelectedTexture);
00098         else
00099             mSprite.setTexture(mNormalTexture);
00100     }
00101 }
00102
00103 void Button::handleEvent(const sf::Event&)
00104 {
00105 }
00106
00107 void Button::draw(sf::RenderTarget& target, sf::RenderStates states) const
00108 {
00109     states.transform *= getTransform();
00110     target.draw(mSprite, states);
00111     target.draw(mText, states);
00112 }
00113
00114 }
```

6.7 Source/Character.cpp File Reference

```
#include <Character.hpp>
#include <ResourceHolder.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <CharacterState.hpp>
#include <cmath>
#include <iostream>
#include <set>
```

Include dependency graph for Character.cpp:

Functions

- Textures::ID toTextureIDMoving (Character::Type type)

- Textures::ID toTextureIDDeath (Character::Type type)
- Character::Type setType (TypeCharacter::ID type)
- int getSizeFrame (Character::Type type)

6.7.1 Function Documentation

6.7.1.1 getSizeFrame()

```
int getSizeFrame (
            Character::Type type )
```

Definition at line 77 of file Character.cpp.

6.7.1.2 setType()

```
Character::Type setType (
            TypeCharacter::ID type )
```

Definition at line 55 of file Character.cpp.

6.7.1.3 toTextureIDDeath()

```
Textures::ID toTextureIDDeath (
            Character::Type type )
```

Definition at line 34 of file Character.cpp.

6.7.1.4 toTextureIDMoving()

```
Textures::ID toTextureIDMoving (
            Character::Type type )
```

Definition at line 13 of file Character.cpp.

6.8 Character.cpp

Go to the documentation of this file.

```
00001 #include <Character.hpp>
00002 #include <ResourceHolder.hpp>
00003
00004 #include <SFML/Graphics/RenderTarget.hpp>
00005 #include <SFML/Graphics/RenderStates.hpp>
00006 #include <SFML/Graphics/Texture.hpp>
00007 #include <CharacterState.hpp>
00008
00009 #include <cmath>
00010 #include <iostream>
00011 #include <set>
00012
00013 Textures::ID toTextureIDMoving(Character::Type type) {
00014     switch (type) {
00015         case Character::BlueDino:
00016             return Textures::BlueDino;
00017         case Character::RedDino:
00018             return Textures::RedDino;
00019         case Character::YellowDino:
00020             return Textures::YellowDino;
00021         case Character::GreenDino:
00022             return Textures::GreenDino;
00023         case Character::BlueFrog:
00024             return Textures::BlueFrog;
00025         case Character::GreenFrog:
00026             return Textures::GreenFrog;
00027         case Character::PinkFrog:
00028             return Textures::PinkFrog;
00029         case Character::YellowFrog:
00030             return Textures::YellowFrog;
00031     }
00032     return Textures::Player;
00033 }
00034 Textures::ID toTextureIDDeath(Character::Type type) {
00035     switch (type) {
00036         case Character::BlueDino:
00037             return Textures::BlueDinoDeath;
00038         case Character::RedDino:
00039             return Textures::RedDinoDeath;
00040         case Character::YellowDino:
00041             return Textures::YellowDinoDeath;
00042         case Character::GreenDino:
00043             return Textures::GreenDinoDeath;
00044         case Character::BlueFrog:
00045             return Textures::BlueFrogDeath;
00046         case Character::GreenFrog:
00047             return Textures::GreenFrogDeath;
00048         case Character::PinkFrog:
00049             return Textures::PinkFrogDeath;
00050         case Character::YellowFrog:
00051             return Textures::YellowFrogDeath;
00052     }
00053     return Textures::Player;
00054 }
00055 Character::Type setType(TypeCharacter::ID type){
00056     switch (type) {
00057         case TypeCharacter::BlueDino:
00058             return Character::Type::BlueDino;
00059         case TypeCharacter::RedDino:
00060             return Character::Type::RedDino;
00061         case TypeCharacter::GreenDino:
00062             return Character::Type::GreenDino;
00063         case TypeCharacter::YellowDino:
00064             return Character::Type::YellowDino;
00065         case TypeCharacter::BlueFrog:
00066             return Character::Type::BlueFrog;
00067         case TypeCharacter::GreenFrog:
00068             return Character::Type::GreenFrog;
00069         case TypeCharacter::YellowFrog:
00070             return Character::Type::YellowFrog;
00071         case TypeCharacter::PinkFrog:
00072             return Character::Type::PinkFrog;
00073         default:
00074             throw "Not found type characrer";
00075     }
00076 }
00077 int getSizeFrame(Character::Type type){
00078     switch (type)
00079     {
00080     case Character::BlueDino:
00081         return 24;
00082     case Character::RedDino:
```

```
00083         return 24;
00084     case Character::YellowDino:
00085         return 24;
00086     case Character::GreenDino:
00087         return 24;
00088     case Character::BlueFrog:
00089         return 16;
00090     case Character::YellowFrog:
00091         return 16;
00092     case Character::PinkFrog:
00093         return 16;
00094     case Character:: GreenFrog:
00095         return 16;
00096     }
00097     throw "Not found type characrer";
00098 }
00099 Character::Character(Type type, const TextureHolder& textures) :
      mSprite(textures.get(Textures::Player)) {
00100
00101     mType = setType(typeCharacter);
00102     mDeath.setTexture(textures.get(toTextureIDDeath(mType)));
00103     mMoving.setTexture(textures.get(toTextureIDMoving(mType)));
00104
00105     health=Constants::characterHealth;
00106
00107     int framSize = getSizeFrame(mType);
00108     mDeath.setFrameSize(sf::Vector2i(framSize, framSize));
00109     mDeath.setNumFrames(8);
00110     mDeath.setDuration(sf::seconds(1));
00111
00112     mMoving.setFrameSize(sf::Vector2i(framSize, framSize));
00113     mMoving.setNumFrames(8);
00114     mMoving.setDuration(sf::seconds(0.2));
00115
00116     mDeath.setOrigin(mDeath.getLocalBounds().width / 2.f, mDeath.getLocalBounds().height / 2.f);
00117     mDeath.scale(mStep / mDeath.getLocalBounds().width, mStep / mDeath.getLocalBounds().height);
00118     mMoving.setOrigin(mMoving.getLocalBounds().width / 2.f, mMoving.getLocalBounds().height / 2.f);
00119     mMoving.scale(mStep / mMoving.getLocalBounds().width, mStep / mMoving.getLocalBounds().height);
00120
00121     mSprite.scale(mStep / mSprite.getLocalBounds().width, mStep / mSprite.getLocalBounds().height);
00122     sf::FloatRect bounds = mSprite.getLocalBounds();
00123     mSprite.setOrigin(bounds.width / 2.f, bounds.height / 2.f);
00124
00125 }
00126
00127 void Character::drawCurrent(sf::RenderTarget& target, sf::RenderStates states) const {
00128     if (mFacing == Facing::Left) {
00129         states.transform.scale(-1.f, 1.f);
00130     }
00131     if (isDestroyed() && !mIsMoving) target.draw(mDeath, states);
00132     else {
00133         target.draw(mMoving, states);
00134         // target.draw(mSprite, states);
00135
00136     }
00137 }
00138
00139 unsigned int Character::getCategory() const {
00140     switch (mType) {
00141         case Player:
00142             return Category::PlayerCharacter;
00143         default:
00144             return Category::PlayerCharacter;
00145     }
00146 }
00147
00148 void Character::pathRequest(sf::Vector2f direction) {
00149     if (mPath.size() < 2)
00150         mPath.push(direction);
00151 }
00152
00153 void Character::updateCurrent(sf::Time dt) {
00154     if (isDestroyed() && !mIsMoving){
00155         mDeath.update(dt);
00156         return;
00157     }
00158     if (!mIsMoving) {;
00159         mGridPosition = getPosition();
00160         if (!mPath.empty()) {
00161             sf::Vector2f direction = mPath.front();
00162             mPath.pop();
00163             if (!predictMovement(direction)) {
00164                 return;
00165             }
00166             mMovement = direction;
00167             mIsMoving = true;
00168             if (mMovement.x > 0) {
```

```
00169                 mFacing = Facing::Right;
00170             }
00171             else if (mMovement.x < 0) {
00172                 mFacing = Facing::Left;
00173             }
00174         }
00175     }
00176     mMoving.update(dt);
00177     if (mIsMoving) {
00178         if (mMoving.isFinished()){
00179             mMoving.restart();
00180             return;
00181         }
00182         sf::Vector2f movement = mMovement * dt.asSeconds() * speedMult;
00183         if (mDistanceTravelled + abs(movement.x) > mStep) {
00184             mDistanceTravelled = 0.f;
00185             mIsMoving = false;
00186             if (movement.x > 0) {
00187                 mGridPosition.x += mStep;
00188             }
00189             else {
00190                 mGridPosition.x -= mStep;
00191             }
00192             setPosition(mGridPosition);
00193         }
00194         else if (mDistanceTravelled + abs(movement.y) > mStep) {
00195             mDistanceTravelled = 0.f;
00196             mIsMoving = false;
00197             if (movement.y > 0) {
00198                 mGridPosition.y += mStep;
00199             }
00200             else {
00201                 mGridPosition.y -= mStep;
00202             }
00203             setPosition(mGridPosition);
00204         }
00205         else {
00206             mDistanceTravelled += abs(movement.x) + abs(movement.y);
00207             move(movement);
00208         }
00209     }
00210     updateTemperature(dt);
00211     updateSpeedMult(dt);
00212     updateHealth(dt);
00213 }
00214 sf::FloatRect Character::getBoundingRect() const
00215 {
00216     return getWorldTransform().transformRect(mSprite.getGlobalBounds());
00217 }
00218
00219 bool Character::isMarkedForRemoval() const
00220 {
00221     //return isDestroyed() && (mExplosion.isFinished() || !mShowExplosion);
00222     // return false;
00223     return isDestroyed() && (mDeath.isFinished() || !mShowDeath);
00224     return false;
00225     return isDestroyed();
00226 }
00227
00228 void Character::setWorldSceneGraph(SceneNode* worldSceneGraph) {
00229     this->worldSceneGraph = worldSceneGraph;
00230 }
00231
00232 bool Character::predictMovement(sf::Vector2f direction) {
00233     sf::FloatRect bounds = getBoundingRect();
00234     if (direction.x < 0) {
00235         bounds.left -= mStep;
00236     }
00237     else if (direction.x > 0) {
00238         bounds.left += mStep;
00239     }
00240     else if (direction.y < 0) {
00241         bounds.top -= mStep;
00242     }
00243     else if (direction.y > 0) {
00244         bounds.top += mStep;
00245     }
00246
00247     std::set<SceneNode*> collidingNodes;
00248     worldSceneGraph->checkNodeCollision(bounds, collidingNodes);
00249
00250     auto matchesCategories = [](SceneNode* node, Category::Type type) {
00251         return (node->getCategory() & type) != 0;
00252     };
00253
00254     for (auto node : collidingNodes) {
00255         if (matchesCategories(node, Category::Stone)) {
```

```
00256             return false;
00257         }
00258     }
00259     return true;
00260 }
00261
00262 void Character::destroy() {
00263     // std::cout « "Character destroyed\n";
00264     if (!mIsDestroyed) {
00265         playerSoundController.play(SoundEffects::Die);
00266         mIsDestroyed = true;
00267         mShowDeath = true;
00268         mDeath.restart();
00269     }
00270 }
00271
00272 bool Character::isDestroyed() const {
00273     return mIsDestroyed|isDestroyedFlag;
00274 }
00275
00276 void Character::updateRollAnimation(){
00277 }
00278
00279 void Character::setFreezing() {
00280     isCold=true;
00281 }
00282
00283 void Character::notFreezing() {
00284     isCold=false;
00285 }
00286
00287 bool Character::isFreezing() {
00288     return getTemperature()<Constants::freezeLimit;
00289 }
00290
00291 bool Character::isBurning() {
00292     return getTemperature()>Constants::burningLimit;
00293 }
00294
00295 float Character::getTemperature() {
00296     return temperature;
00297 }
00298
00299 void Character::shiftTemperature(float offset) {
00300     temperature+=offset;
00301 }
00302
00303 void Character::setTemperature(float value) {
00304     temperature=value;
00305 }
00306
00307 void Character::setDefaultTemperature(float value) {
00308     defaultTemperature=value;
00309 }
00310
00311 void Character::updateTemperature(sf::Time dt) {
00312
      temperature-=(temperature-defaultTemperature)*(dt.asMilliseconds()/(dt.asMilliseconds()+Constants::TemperatureSlope));
00313 }
00314
00315 float Character::getSpeedMult() {
00316     return speedMult;
00317 }
00318
00319 void Character::shiftSpeedMult(float offset) {
00320     speedMult+=offset;
00321 }
00322
00323 void Character::multSpeedMult(float offset) {
00324     speedMult*=offset;
00325 }
00326
00327 void Character::setSpeedMult(float value) {
00328     speedMult=value;
00329 }
00330
00331 void Character::setDefaultSpeedMult(float value) {
00332     defaultSpeedMult=value;
00333 }
00334
00335 void Character::updateSpeedMult(sf::Time dt) {
00336     if (isFreezing()) {
00337         setDefaultSpeedMult(Constants::FreezingDefaultSpeed);
00338     }
00339     else if (isBurning()) {
00340         setDefaultSpeedMult(Constants::BurningDefaultSpeed);
00341     }
```

```
00342     else {
00343         setDefaultSpeedMult(1);
00344     }
00345
      speedMult-=(speedMult-defaultSpeedMult)*(dt.asMilliseconds()/(dt.asMilliseconds()+Constants::SpeedSlope));
00346 }
00347
00348 float Character::getHealth() {
00349     return health;
00350 }
00351
00352 void Character::hurt(float x) {
00353     if (x>0&&x==Constants::hurtAmountSmall) {
00354         playerSoundController.play(SoundEffects::Hurt);
00355     }
00356     if (x>0&&x==Constants::hurtAmountLarge) {
00357         playerSoundController.play(SoundEffects::Hurt);
00358     }
00359     if (x<0&&x==-Constants::healAmountSmall) {
00360         playerSoundController.play(SoundEffects::Heal);
00361     }
00362     if (x<0&&x==-Constants::healAmountLarge) {
00363         playerSoundController.play(SoundEffects::Heal);
00364     }
00365     health-=x;
00366     if (health>Constants::characterHealth) {
00367         health=Constants::characterHealth;
00368     }
00369 }
00370
00371 void Character::updateHealth(sf::Time dt) {
00372     if (isFreezing()) {
00373         hurt(Constants::FreezingHealthDrainPerSecond*dt.asSeconds());
00374     }
00375     if (isBurning()) {
00376         hurt(Constants::BurningHealthDrainPerSecond*dt.asSeconds());
00377     }
00378 }
```

6.9 Source/CharacterState.cpp File Reference

```
#include <CharacterState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <iostream>
#include <Const.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Image.hpp>
#include <SFML/System/Clock.hpp>
#include <SFML/System/Time.hpp>
```

Include dependency graph for CharacterState.cpp:

Functions

- TypeCharacter::ID setTypeCha (int type)

Variables

- TypeCharacter::ID typeCharacter

6.9.1 Function Documentation

6.9.1.1 setTypeCha()

```
TypeCharacter::ID setTypeCha (
            int type )
```

Definition at line 13 of file CharacterState.cpp.

6.9.2 Variable Documentation

6.9.2.1 typeCharacter

```
TypeCharacter::ID typeCharacter
```

Definition at line 11 of file CharacterState.cpp.

6.10 CharacterState.cpp

Go to the documentation of this file.

```
00001 #include <CharacterState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004 #include <iostream>
00005 #include <Const.hpp>
00006
00007 #include <SFML/Graphics/RenderWindow.hpp>
00008 #include <SFML/Graphics/Image.hpp>
00009 #include <SFML/System/Clock.hpp>
00010 #include <SFML/System/Time.hpp>
00011 TypeCharacter::ID typeCharacter;
00012
00013 TypeCharacter::ID setTypeCha(int type){
00014     switch (type) {
00015     case 0:
00016         return TypeCharacter::BlueDino;
00017     case 1:
00018         return TypeCharacter::GreenDino;
00019     case 2:
00020         return TypeCharacter::YellowDino;
00021     case 3:
00022         return TypeCharacter::RedDino;
00023     case 4:
00024         return TypeCharacter::BlueFrog;
00025     case 5:
00026         return TypeCharacter::GreenFrog;
00027     case 6:
00028         return TypeCharacter::YellowFrog;
00029     case 7:
00030         return TypeCharacter::PinkFrog;
00031     default:
00032         throw "Do not find corresponding character!";
00033     }
00034 }
00035 CharacterState::CharacterState(StateStack &stack, Context context)
00036 : State(stack, context)
00037 , mGUIContainer()
00038 {
00039     mBackgroundSprite.setTexture(context.textures->get(Textures::Character));
00040
00041     addCharacterTexture("Dinosaur", "Blue", 24);
00042     addBackGroudCharacterTexture("Dinosaur", "Blue");
00043
00044     addCharacterTexture("Dinosaur", "Green", 24);
00045     addBackGroudCharacterTexture("Dinosaur", "Green");
00046
00047     addCharacterTexture("Dinosaur", "Yellow", 24);
00048     addBackGroudCharacterTexture("Dinosaur", "Yellow");
00049
00050     addCharacterTexture("Dinosaur", "Red", 24);
00051     addBackGroudCharacterTexture("Dinosaur", "Red");
00052
00053     addCharacterTexture("Frog", "Blue", 11);
00054     addBackGroudCharacterTexture("Frog", "Blue");
00055
00056     addCharacterTexture("Frog", "Green", 11);
00057     addBackGroudCharacterTexture("Frog", "Green");
00058
00059     addCharacterTexture("Frog", "Yellow", 11);
00060     addBackGroudCharacterTexture("Frog", "Yellow");
00061
00062     addCharacterTexture("Frog", "Pink", 11);
00063     addBackGroudCharacterTexture("Frog", "Pink");
00064
00065     mName.setFillColor(sf::Color::Black);
00066     mName.setCharacterSize(40);
```

```
00067     mName.setOutlineThickness(0.5);
00068     mName.setFont(context.fonts->get(Fonts::Main));
00069     mName.setPosition(830, 200);
00070     mName.setString(listName[0]);
00071 
00072     mCharacterSprite.setTexture(mCharacterTexture[0][0]);
00073     mCharacterSprite.setPosition(830, 500);
00074 
00075     mBackgroundSpriteCharacter.setTexture(mBackgroundTextureCharacter[0]);
00076     mBackgroundSpriteCharacter.setScale(0.9f, 0.9f);
00077     mBackgroundSpriteCharacter.setPosition(740, 175);
00078 
00079     auto backButton = std::make_shared<GUI::Button>(context);
00080     backButton->setPosition(70.f, 950.f);
00081     backButton->setText("Back", 40);
00082     backButton->setCallback([this]()
00083     {
00084         requestStackPop();
00085         requestStackPush(States::Menu);
00086     });
00087 
00088     auto playButton = std::make_shared<GUI::Button>(context);
00089     playButton->setPosition(1540.f, 950.f);
00090     playButton->setText("Next", 40);
00091     playButton->setCallback([this]()
00092     {
00093         requestStackPop();
00094         requestStackPush(States::Map);
00095     });
00096     mGUIContainer.pack(backButton);
00097     mGUIContainer.pack(playButton);
00098 }
00099 void CharacterState::addCharacterTexture(const std::string &nameCharacter, const std::string &color,
      int numOfPicture){
00100     std::string nameFile = color + nameCharacter;
00101     listName.push_back(color + " " + nameCharacter);
00102     std::vector<sf::Texture> frame;
00103     sf::Texture texture;
00104     for (int i = 1; i <= numOfPicture; i++){
00105         if (!texture.loadFromFile("Media/Textures/Characters/" + nameCharacter + "/" + nameFile + "_"
      + std::to_string(i) + ".png")) {
00106             std::cerr « "Error loading image frame" « i « ".png" « std::endl;
00107             return;
00108         }
00109 
00110         frame.push_back(resizeTexture(texture, sf::Vector2u(240, 240)));
00111     }
00112     mCharacterTexture.push_back(frame);
00113     return;
00114 }
00115 void CharacterState::addBackGroudCharacterTexture(const std::string &nameCharacter, const std::string
      &color){
00116     std::string nameFile = color + nameCharacter;
00117     sf::Texture texture;
00118     if (!texture.loadFromFile("Media/Textures/Characters/" + nameCharacter + "/" + nameFile +
      "BG.png")) {
00119         std::cerr « "Error loading BG image frame of " « nameFile « ".png" « std::endl;
00120         return;
00121     }
00122     mBackgroundTextureCharacter.push_back(texture);
00123     return;
00124 }
00125 void CharacterState::draw(){
00126     sf::RenderWindow& window = *getContext().window;
00127     window.setView(window.getDefaultView());
00128 
00129     window.draw(mBackgroundSprite);
00130     window.draw(mBackgroundSpriteCharacter);
00131     window.draw(mName);
00132     window.draw(mCharacterSprite);
00133     window.draw(mGUIContainer);
00134 }
00135 bool CharacterState::handleEvent(const sf::Event& event){
00136     if (event.type == sf::Event::KeyPressed){
00137         if (event.key.code == sf::Keyboard::D || event.key.code == sf::Keyboard::Right){
00138             if (currentType < mBackgroundTextureCharacter.size() - 1){
00139                 currentType++;
00140             }
00141         }
00142         else if (event.key.code == sf::Keyboard::A || event.key.code == sf::Keyboard::Left){
00143             if (currentType > 0){
00144                 currentType--;
00145             }
00146         }
00147     }
00148     mGUIContainer.handleEvent(event);
00149     return true;
```

```
00150 }
00151 bool CharacterState::update(sf::Time dt){
00152     if (clock.getElapsedTime() >= frameTime){
00153 
00154         mName.setString(listName[currentType]);
00155         mCharacterSprite.setTexture(mCharacterTexture[currentType][currentFrame]);
00156         if (++currentFrame >= mCharacterTexture[currentType].size()) currentFrame = 0;
00157         clock.restart();
00158     }
00159     mBackgroundSpriteCharacter.setTexture(mBackgroundTextureCharacter[currentType]);
00160 
00161     sf::FloatRect textBounds = mName.getLocalBounds();
00162     mName.setOrigin(textBounds.left + textBounds.width / 2.0f, textBounds.top + textBounds.height /
    2.0f);
00163     mName.setPosition(mBackgroundSpriteCharacter.getPosition().x +
    mBackgroundSpriteCharacter.getGlobalBounds().width / 2.0f,
00164     mBackgroundSpriteCharacter.getPosition().y + mBackgroundSpriteCharacter.getGlobalBounds().height /
    2.0f - 200);
00165 
00166     typeCharacter = setTypeCha(currentType);
00167 
00168     return true;
00169 }
00170 //Function resize the texture
00171 sf::Texture CharacterState::resizeTexture(const sf::Texture& originalTexture, const sf::Vector2u&
    targetSize) {
00172 
00173     sf::Image originalImage = originalTexture.copyToImage();
00174     sf::Image resizedImage;
00175     resizedImage.create(targetSize.x, targetSize.y, sf::Color::Transparent);
00176 
00177     float xScale = static_cast<float>(targetSize.x) / originalImage.getSize().x;
00178     float yScale = static_cast<float>(targetSize.y) / originalImage.getSize().y;
00179 
00180     for (unsigned int x = 0; x < targetSize.x; ++x) {
00181         for (unsigned int y = 0; y < targetSize.y; ++y) {
00182             unsigned int origX = static_cast<unsigned int>(x / xScale);
00183             unsigned int origY = static_cast<unsigned int>(y / yScale);
00184 
00185             sf::Color pixelColor = originalImage.getPixel(origX, origY);
00186             resizedImage.setPixel(x, y, pixelColor);
00187         }
00188     }
00189 
00190     sf::Texture resizedTexture;
00191     resizedTexture.loadFromImage(resizedImage);
00192     resizedTexture.setSmooth(false);
00193     return resizedTexture;
00194 }
```

6.11 Source/Command.cpp File Reference

`#include <Command.hpp>`
Include dependency graph for Command.cpp:

6.12 Command.cpp

Go to the documentation of this file.

```
00001 #include <Command.hpp>
00002 
00003 Command::Command() : action(), category(Category::None) {}
```

6.13 Source/CommandQueue.cpp File Reference

`#include <CommandQueue.hpp>`
`#include <SceneNode.hpp>`
Include dependency graph for CommandQueue.cpp:

6.14 CommandQueue.cpp

Go to the documentation of this file.

```
00001 #include <CommandQueue.hpp>
00002 #include <SceneNode.hpp>
00003
00004 void CommandQueue::push(const Command& command) {
00005     mQueue.push(command);
00006 }
00007
00008 Command CommandQueue::pop() {
00009     Command command = mQueue.front();
00010     mQueue.pop();
00011     return command;
00012 }
00013
00014 bool CommandQueue::isEmpty() const {
00015     return mQueue.empty();
00016 }
```

6.15 Source/Component.cpp File Reference

`#include <Component.hpp>`
Include dependency graph for Component.cpp:

Namespaces

- namespace GUI

6.16 Component.cpp

Go to the documentation of this file.

```
00001 #include <Component.hpp>
00002 namespace GUI
00003 {
00004 Component::Component()
00005 : mIsSelected(false)
00006 , mIsActive(false)
00007 {
00008 }
00009 Component::~Component()
00010 {
00011 }
00012
00013 bool Component::isSelected() const
00014 {
00015     return mIsSelected;
00016 }
00017
00018 void Component::select()
00019 {
00020     mIsSelected = true;
00021 }
00022
00023 void Component::deselect()
00024 {
00025     mIsSelected = false;
00026 }
00027
00028 bool Component::isActive() const
00029 {
00030     return mIsActive;
00031 }
00032
00033 void Component::activate()
00034 {
00035     mIsActive = true;
00036 }
00037
00038 void Component::deactivate()
00039 {
00040     mIsActive = false;
00041 }
00042
00043 }
```

6.17 Source/Container.cpp File Reference

```
#include <Container.hpp>
#include <Foreach.hpp>
#include <SFML/Window/Event.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for Container.cpp:

Namespaces

- namespace GUI

6.18 Container.cpp

Go to the documentation of this file.

```
00001 #include <Container.hpp>
00002 #include <Foreach.hpp>
00003
00004 #include <SFML/Window/Event.hpp>
00005 #include <SFML/Graphics/RenderStates.hpp>
00006 #include <SFML/Graphics/RenderTarget.hpp>
00007
00008
00009 namespace GUI
00010 {
00011
00012 Container::Container()
00013 : mChildren()
00014 , mSelectedChild(-1)
00015 {
00016 }
00017 void Container::pack(Component::Ptr component)
00018 {
00019     mChildren.push_back(component);
00020
00021     if (!hasSelection() && component->isSelectable())
00022         select(mChildren.size() - 1);
00023 }
00024
00025 bool Container::isSelectable() const
00026 {
00027     return false;
00028 }
00029
00030 void Container::handleEvent(const sf::Event& event)
00031 {
00032     // If we have selected a child then give it events
00033     if (hasSelection() && mChildren[mSelectedChild]->isActive())
00034     {
00035         mChildren[mSelectedChild]->handleEvent(event);
00036     }
00037     else if (event.type == sf::Event::KeyPressed)
00038     {
00039         if (event.key.code == sf::Keyboard::W || event.key.code == sf::Keyboard::Up)
00040         {
00041             selectPrevious();
00042         }
00043         else if (event.key.code == sf::Keyboard::S || event.key.code == sf::Keyboard::Down)
00044         {
00045             selectNext();
00046         }
00047         else if (event.key.code == sf::Keyboard::Return || event.key.code == sf::Keyboard::Space)
00048         {
00049             if (hasSelection())
00050                 mChildren[mSelectedChild]->activate();
00051         }
00052     }
00053 }
00054
00055 void Container::clear() {
00056     mChildren.clear();
00057 }
```

```
00058
00059 void Container::draw(sf::RenderTarget& target, sf::RenderStates states) const
00060 {
00061     states.transform *= getTransform();
00062
00063     FOREACH(const Component::Ptr& child, mChildren)
00064         target.draw(*child, states);
00065 }
00066
00067 bool Container::hasSelection() const
00068 {
00069     return mSelectedChild >= 0;
00070 }
00071
00072 void Container::select(std::size_t index)
00073 {
00074     if (mChildren[index]->isSelectable())
00075     {
00076         if (hasSelection())
00077             mChildren[mSelectedChild]->deselect();
00078
00079         mChildren[index]->select();
00080         mSelectedChild = index;
00081     }
00082 }
00083
00084 void Container::selectNext()
00085 {
00086     if (!hasSelection())
00087         return;
00088
00089     // Search next component that is selectable, wrap around if necessary
00090     int next = mSelectedChild;
00091     do
00092         next = (next + 1) % mChildren.size();
00093     while (!mChildren[next]->isSelectable());
00094
00095     // Select that component
00096     select(next);
00097 }
00098
00099 void Container::selectPrevious()
00100 {
00101     if (!hasSelection())
00102         return;
00103
00104     // Search previous component that is selectable, wrap around if necessary
00105     int prev = mSelectedChild;
00106     do
00107         prev = (prev + mChildren.size() - 1) % mChildren.size();
00108     while (!mChildren[prev]->isSelectable());
00109
00110     // Select that component
00111     select(prev);
00112 }
00113
00114 }
```

6.19 Source/CountDownState.cpp File Reference

```
#include <CountDownState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <MusicPlayer.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
#include <cmath>
```

Include dependency graph for CountDownState.cpp:

6.20 CountDownState.cpp

Go to the documentation of this file.

```
00001 #include <CountDownState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004 #include <MusicPlayer.hpp>
00005
00006 #include <SFML/Graphics/RectangleShape.hpp>
00007 #include <SFML/Graphics/RenderWindow.hpp>
00008 #include <SFML/Graphics/View.hpp>
00009 #include <cmath>
00010
00011 CountDownState::CountDownState(StateStack& stack, Context context)
00012     : State(stack, context)
00013     , mCountdownText()
00014     , mCountdownTime(sf::seconds(4))
00015     , mElapsedTime(sf::Time::Zero)
00016 {
00017     mBackground.setTexture(context.textures->get(Textures::Countdown));
00018
00019     sf::Font& font = context.fonts->get(Fonts::Main);
00020     sf::Vector2f windowSize(context.window->getSize());
00021
00022     mCountdownText.setFont(font);
00023     mCountdownText.setCharacterSize(200);
00024
00025     // Customize the appearance of the countdown text
00026     mCountdownText.setFillColor(sf::Color::White);
00027     centerOrigin(mCountdownText);
00028     mCountdownText.setPosition(0.48f * windowSize.x, 0.35f * windowSize.y);
00029
00030     context.music->play(Music::CountDownTheme);
00031 }
00032
00033 bool CountDownState::update(sf::Time dt)
00034 {
00035     if (mCountdownTime > sf::Time::Zero)
00036     {
00037         mCountdownTime -= dt;
00038         updateCountdownUI();
00039
00040         // If countdown reaches zero, start the game
00041         if (mCountdownTime <= sf::Time::Zero)
00042         {
00043             requestStackPop();
00044             requestStackPush(States::Game);
00045         }
00046     }
00047
00048     return false;
00049 }
00050
00051 void CountDownState::draw()
00052 {
00053     sf::RenderWindow& window = *getContext().window;
00054     window.setView(window.getDefaultView());
00055
00056
00057     window.draw(mBackground);
00058     window.draw(mCountdownText);
00059 }
00060
00061 bool CountDownState::handleEvent(const sf::Event&)
00062 {
00063     return false;
00064 }
00065
00066 void CountDownState::updateCountdownUI()
00067 {
00068     // Calculate the remaining seconds in the countdown
00069     int seconds = static_cast<int>(std::ceil(mCountdownTime.asSeconds()));
00070
00071     // Update the UI to display the countdown
00072     updateUIWithCountdown(seconds);
00073 }
00074
00075 void CountDownState::updateUIWithCountdown(int seconds)
00076 {
00077     // Clear previous UI
00078     clearUI();
00079
00080     // Display the countdown
00081     if (seconds > 1)
00082     {
00083         // Draw the countdown on the screen
00084         drawCountdownText(seconds);
00085     }
00086     else
00087     {
```

```
00088         //Signal the start
00089         updateUIForGo();
00090     }
00091 }
00092
00093 void CountDownState::drawCountdownText(int seconds)
00094 {
00095     // Set the text to display the countdown
00096     mCountdownText.setString(std::to_string(seconds-1));
00097
00098     // Draw the countdown text
00099     sf::RenderWindow& window = *getContext().window;
00100     window.draw(mCountdownText);
00101 }
00102
00103 void CountDownState::clearUI()
00104 {
00105     sf::RenderWindow& window = *getContext().window;
00106     window.clear();
00107 }
00108
00109 void CountDownState::updateUIForGo()
00110 {
00111     mCountdownText.setString("GO");
00112     centerOrigin(mCountdownText);
00113
00114     sf::Vector2f windowSize = getContext().window->getView().getSize();
00115
00116     mCountdownText.setPosition(0.5f * windowSize.x, 0.48f * windowSize.y);
00117
00118     sf::RenderWindow& window = *getContext().window;
00119     window.draw(mCountdownText);
00120 }
```

6.21 Source/CreditState.cpp File Reference

```
#include <CreditState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
```

Include dependency graph for CreditState.cpp:

6.22 CreditState.cpp

Go to the documentation of this file.

```
00001 #include <CreditState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004
00005 #include <SFML/Graphics/RenderWindow.hpp>
00006
00007 CreditState::CreditState(StateStack& stack, Context context)
00008 : State(stack, context)
00009 , mGUIContainer()
00010 {
00011     mBackgroundSprite.setTexture(context.textures->get(Textures::Credit));
00012
00013     // Create a text object for the introduction line
00014     mIntroductionText.setFont(context.fonts->get(Fonts::Main));
00015     mIntroductionText.setCharacterSize(60);
00016     mIntroductionText.setFillColor(sf::Color::Black);
00017     mIntroductionText.setString("Our Team");
00018     mIntroductionText.setPosition(850, 250);
00019
00020     for (int i = 0; i < 5; i++){
00021         mCredit[i].setFillColor(sf::Color::Black);
00022         mCredit[i].setFont(context.fonts->get(Fonts::Main));
00023         mCredit[i].setCharacterSize(40);
00024         mCredit[i].setPosition(800, 400 + 100*i);
00025     }
00026     mCredit[0].setString("Ngo Hoang Bao Thach");
00027     mCredit[1].setString("Nguyen Minh Luan");
00028     mCredit[2].setString("Tran Nhat Thanh");
```

```
00029     mCredit[3].setString("Le Van Cuong");
00030     mCredit[4].setString("Vu Hoang Tung");
00031
00032     auto backButton = std::make_shared<GUI::Button>(context);
00033     backButton->setPosition(70.f, 970.f);
00034     backButton->setText("Back", 40);
00035     backButton->setCallback([this]()
00036     {
00037         requestStackPop();
00038     });
00039     mGUIContainer.pack(backButton);
00040 }
00041
00042 void CreditState::draw()
00043 {
00044     sf::RenderWindow& window = *getContext().window;
00045
00046     window.draw(mBackgroundSprite);
00047     window.draw(mIntroductionText);
00048     for (int i = 0; i < 5; i++) window.draw(mCredit[i]);
00049     window.draw(mGUIContainer);
00050 }
00051
00052 bool CreditState::update(sf::Time)
00053 {
00054     return false;
00055 }
00056
00057 bool CreditState::handleEvent(const sf::Event& event)
00058 {
00059     mGUIContainer.handleEvent(event);
00060     return false;
00061 }
```

6.23 Source/DataTables.cpp File Reference

`#include <DataTables.hpp>`
Include dependency graph for DataTables.cpp:

Functions

- std::vector< ObstacleData > initializeObstacleData ()

6.23.1 Function Documentation

6.23.1.1 initializeObstacleData()

`std::vector< ObstacleData > initializeObstacleData ( )`

Definition at line 6 of file DataTables.cpp.

6.24 DataTables.cpp

Go to the documentation of this file.

```
00001 #include <DataTables.hpp>
00002
00003 // For std::bind() placeholders _1, _2, ...
00004 using namespace std::placeholders;
00005
00006 std::vector<ObstacleData> initializeObstacleData()
00007 {
00008     std::vector<ObstacleData> data(Obstacle::Type::TypeCount);
00009
```

```
00010     data[Obstacle::Type::Car].speed = sf::Vector2f(Constants::carSpeed, 0.f);
00011     data[Obstacle::Type::Car].scaleX = false;
00012     data[Obstacle::Type::Car].scaleY = true;
00013     data[Obstacle::Type::Car].texture = Textures::ID::Car;
00014     data[Obstacle::Type::Car].minTime = sf::seconds(1.5);
00015     data[Obstacle::Type::Car].maxTime = sf::seconds(2);
00016     data[Obstacle::Type::Car].groupDelayTime = sf::seconds(5);
00017     data[Obstacle::Type::Car].groupSpawnAmount = 2;
00018     data[Obstacle::Type::Car].spawnOffset = 700.f;
00019     data[Obstacle::Type::Car].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00020
00021     data[Obstacle::Type::Car1].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00022     data[Obstacle::Type::Car1].scaleX = false;
00023     data[Obstacle::Type::Car1].scaleY = true;
00024     data[Obstacle::Type::Car1].texture = Textures::ID::Car;
00025     data[Obstacle::Type::Car1].minTime = sf::seconds(1.6);
00026     data[Obstacle::Type::Car1].maxTime = sf::seconds(2.8);
00027     data[Obstacle::Type::Car1].flipHorizontal = true;
00028     data[Obstacle::Type::Car1].groupDelayTime = sf::seconds(5);
00029     data[Obstacle::Type::Car1].groupSpawnAmount = 3;
00030     data[Obstacle::Type::Car1].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00031
00032     data[Obstacle::Type::Oto].speed = sf::Vector2f(Constants::carSpeed, 0.f);
00033     data[Obstacle::Type::Oto].scaleX = false;
00034     data[Obstacle::Type::Oto].scaleY = true;
00035     data[Obstacle::Type::Oto].texture = Textures::ID::Oto;
00036     data[Obstacle::Type::Oto].minTime = sf::seconds(1.5);
00037     data[Obstacle::Type::Oto].maxTime = sf::seconds(2);
00038     data[Obstacle::Type::Oto].groupDelayTime = sf::seconds(5);
00039     data[Obstacle::Type::Oto].groupSpawnAmount = 2;
00040     data[Obstacle::Type::Oto].spawnOffset = 700.f;
00041     data[Obstacle::Type::Oto].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00042
00043     data[Obstacle::Type::Oto_1].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00044     data[Obstacle::Type::Oto_1].scaleX = false;
00045     data[Obstacle::Type::Oto_1].scaleY = true;
00046     data[Obstacle::Type::Oto_1].texture = Textures::ID::Oto;
00047     data[Obstacle::Type::Oto_1].minTime = sf::seconds(1.6);
00048     data[Obstacle::Type::Oto_1].maxTime = sf::seconds(2.8);
00049     data[Obstacle::Type::Oto_1].flipHorizontal = true;
00050     data[Obstacle::Type::Oto_1].groupDelayTime = sf::seconds(5);
00051     data[Obstacle::Type::Oto_1].groupSpawnAmount = 3;
00052     data[Obstacle::Type::Oto_1].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00053
00054     data[Obstacle::Type::Oto1].speed = sf::Vector2f(Constants::carSpeed, 0.f);
00055     data[Obstacle::Type::Oto1].scaleX = false;
00056     data[Obstacle::Type::Oto1].scaleY = true;
00057     data[Obstacle::Type::Oto1].texture = Textures::ID::Oto1;
00058     data[Obstacle::Type::Oto1].minTime = sf::seconds(1.5);
00059     data[Obstacle::Type::Oto1].maxTime = sf::seconds(2);
00060     data[Obstacle::Type::Oto1].groupDelayTime = sf::seconds(5);
00061     data[Obstacle::Type::Oto1].groupSpawnAmount = 2;
00062     data[Obstacle::Type::Oto1].spawnOffset = 700.f;
00063     data[Obstacle::Type::Oto1].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00064
00065     data[Obstacle::Type::Oto1_1].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00066     data[Obstacle::Type::Oto1_1].scaleX = false;
00067     data[Obstacle::Type::Oto1_1].scaleY = true;
00068     data[Obstacle::Type::Oto1_1].texture = Textures::ID::Oto1;
00069     data[Obstacle::Type::Oto1_1].minTime = sf::seconds(1.6);
00070     data[Obstacle::Type::Oto1_1].maxTime = sf::seconds(2.8);
00071     data[Obstacle::Type::Oto1_1].flipHorizontal = true;
00072     data[Obstacle::Type::Oto1_1].groupDelayTime = sf::seconds(5);
00073     data[Obstacle::Type::Oto1_1].groupSpawnAmount = 3;
00074     data[Obstacle::Type::Oto1_1].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00075
00076     data[Obstacle::Type::Oto2].speed = sf::Vector2f(Constants::carSpeed, 0.f);
00077     data[Obstacle::Type::Oto2].scaleX = false;
00078     data[Obstacle::Type::Oto2].scaleY = true;
00079     data[Obstacle::Type::Oto2].texture = Textures::ID::Oto2;
00080     data[Obstacle::Type::Oto2].minTime = sf::seconds(1.5);
00081     data[Obstacle::Type::Oto2].maxTime = sf::seconds(2);
00082     data[Obstacle::Type::Oto2].groupDelayTime = sf::seconds(5);
00083     data[Obstacle::Type::Oto2].groupSpawnAmount = 2;
00084     data[Obstacle::Type::Oto2].spawnOffset = 700.f;
00085     data[Obstacle::Type::Oto2].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00086
00087     data[Obstacle::Type::Oto2_1].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00088     data[Obstacle::Type::Oto2_1].scaleX = false;
00089
00090     data[Obstacle::Type::Oto2_1].scaleY = true;
00091     data[Obstacle::Type::Oto2_1].texture = Textures::ID::Oto2;
00092     data[Obstacle::Type::Oto2_1].minTime = sf::seconds(1.6);
00093     data[Obstacle::Type::Oto2_1].maxTime = sf::seconds(2.8);
00094     data[Obstacle::Type::Oto2_1].flipHorizontal = true;
00095     data[Obstacle::Type::Oto2_1].groupDelayTime = sf::seconds(5);
00096     data[Obstacle::Type::Oto2_1].groupSpawnAmount = 3;
```

```
00097     data[Obstacle::Type::Oto2_1].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00098
00099     data[Obstacle::Type::Stone].scaleX = false;
00100     data[Obstacle::Type::Stone].scaleY = true;
00101     data[Obstacle::Type::Stone].texture = Textures::ID::Stone;
00102     data[Obstacle::Type::Stone].minDistance = 6;
00103     data[Obstacle::Type::Stone].maxDistance = 10;
00104
00105     data[Obstacle::Type::Tree].scaleX = true;
00106     data[Obstacle::Type::Tree].scaleY = true;
00107     data[Obstacle::Type::Tree].texture = Textures::ID::Tree;
00108     data[Obstacle::Type::Tree].minDistance = 6;
00109     data[Obstacle::Type::Tree].maxDistance = 10;
00110
00111     data[Obstacle::Type::Tree1].scaleX = true;
00112     data[Obstacle::Type::Tree1].scaleY = true;
00113     data[Obstacle::Type::Tree1].texture = Textures::ID::Tree1;
00114     data[Obstacle::Type::Tree1].minDistance = 4;
00115     data[Obstacle::Type::Tree1].maxDistance = 10;
00116
00117     data[Obstacle::Type::Tree2].scaleX = true;
00118     data[Obstacle::Type::Tree2].scaleY = true;
00119     data[Obstacle::Type::Tree2].texture = Textures::ID::Tree2;
00120     data[Obstacle::Type::Tree2].minDistance = 4;
00121     data[Obstacle::Type::Tree2].maxDistance = 10;
00122
00123
00124     data[Obstacle::Type::Tree3].scaleX = true;
00125     data[Obstacle::Type::Tree3].scaleY = true;
00126     data[Obstacle::Type::Tree3].texture = Textures::ID::Tree3;
00127     data[Obstacle::Type::Tree3].minDistance = 4;
00128     data[Obstacle::Type::Tree3].maxDistance = 10;
00129
00130     data[Obstacle::Type::Tree4].scaleX = true;
00131     data[Obstacle::Type::Tree4].scaleY = true;
00132     data[Obstacle::Type::Tree4].texture = Textures::ID::Tree4;
00133     data[Obstacle::Type::Tree4].minDistance = 4;
00134     data[Obstacle::Type::Tree4].maxDistance = 10;
00135
00136     data[Obstacle::Type::Tree5].scaleX = true;
00137     data[Obstacle::Type::Tree5].scaleY = true;
00138     data[Obstacle::Type::Tree5].texture = Textures::ID::Tree5;
00139     data[Obstacle::Type::Tree5].minDistance = 4;
00140     data[Obstacle::Type::Tree5].maxDistance = 10;
00141
00142     data[Obstacle::Type::Island].scaleX = true;
00143     data[Obstacle::Type::Island].scaleY = true;
00144     data[Obstacle::Type::Island].texture = Textures::ID::Island;
00145     data[Obstacle::Type::Island].minDistance = 0;
00146     data[Obstacle::Type::Island].maxDistance = 2;
00147
00148     data[Obstacle::Type::Train].speed = sf::Vector2f(Constants::trainSpeed, 0.f);
00149     data[Obstacle::Type::Train].scaleX = false;
00150     data[Obstacle::Type::Train].scaleY = true;
00151     data[Obstacle::Type::Train].texture = Textures::ID::Train;
00152     data[Obstacle::Type::Train].minTime = sf::seconds(Constants::trainCycleTimeLowerBound);
00153     data[Obstacle::Type::Train].maxTime = sf::seconds(Constants::trainCycleTimeUpperBound);
00154     data[Obstacle::Type::Train].groupDelayTime = sf::seconds(5);
00155     data[Obstacle::Type::Train].groupSpawnAmount = 1;
00156     data[Obstacle::Type::Train].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00157
00158     data[Obstacle::Type::Train1].speed = sf::Vector2f(-Constants::trainSpeed, 0.f);
00159     data[Obstacle::Type::Train1].scaleX = false;
00160     data[Obstacle::Type::Train1].scaleY = true;
00161     data[Obstacle::Type::Train1].texture = Textures::ID::Train;
00162     data[Obstacle::Type::Train1].minTime = sf::seconds(Constants::trainCycleTimeLowerBound);
00163     data[Obstacle::Type::Train1].maxTime = sf::seconds(Constants::trainCycleTimeUpperBound);
00164     data[Obstacle::Type::Train1].flipHorizontal = true;
00165     data[Obstacle::Type::Train1].groupDelayTime = sf::seconds(5);
00166     data[Obstacle::Type::Train1].groupSpawnAmount = 1;
00167     data[Obstacle::Type::Train1].passTime = sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00168
00169     data[Obstacle::Type::TrafficLightRed].scaleX = false;
00170     data[Obstacle::Type::TrafficLightRed].scaleY = true;
00171     data[Obstacle::Type::TrafficLightRed].texture = Textures::ID::TrafficLightRed;
00172
00173     data[Obstacle::Type::TrafficLightGreen].scaleX = false;
00174     data[Obstacle::Type::TrafficLightGreen].scaleY = true;
00175     data[Obstacle::Type::TrafficLightGreen].texture = Textures::ID::TrafficLightGreen;
00176
00177     data[Obstacle::Type::TrafficLightYellow].scaleX = false;
00178     data[Obstacle::Type::TrafficLightYellow].scaleY = true;
00179     data[Obstacle::Type::TrafficLightYellow].texture = Textures::ID::TrafficLightYellow;
00180
00181     data[Obstacle::Type::SpeedUp].scaleX = true;
00182     data[Obstacle::Type::SpeedUp].scaleY = true;
00183     data[Obstacle::Type::SpeedUp].texture = Textures::ID::SpeedUp;
```

```
00184     data[Obstacle::Type::SpeedUp].minDistance = 10;
00185     data[Obstacle::Type::SpeedUp].maxDistance = 40;
00186
00187     data[Obstacle::Type::SlowDown].scaleX = true;
00188     data[Obstacle::Type::SlowDown].scaleY = true;
00189     data[Obstacle::Type::SlowDown].texture = Textures::ID::SlowDown;
00190     data[Obstacle::Type::SlowDown].minDistance = 10;
00191     data[Obstacle::Type::SlowDown].maxDistance = 40;
00192
00193     data[Obstacle::Type::IceCream].scaleX = true;
00194     data[Obstacle::Type::IceCream].scaleY = true;
00195     data[Obstacle::Type::IceCream].texture = Textures::ID::IceCream;
00196     data[Obstacle::Type::IceCream].minDistance = 10;
00197     data[Obstacle::Type::IceCream].maxDistance = 40;
00198
00199     data[Obstacle::Type::Animal1].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00200     data[Obstacle::Type::Animal1].scaleX = false;
00201
00202     data[Obstacle::Type::Animal1].scaleY = false;
00203     data[Obstacle::Type::Animal1].hasAnimation = true;
00204     data[Obstacle::Type::Animal1].animation = Animations::ID::Animal1;
00205     data[Obstacle::Type::Animal1].texture = Textures::ID::Oto2;
00206     data[Obstacle::Type::Animal1].minTime = sf::seconds(1.6);
00207     data[Obstacle::Type::Animal1].maxTime = sf::seconds(2.8);
00208     data[Obstacle::Type::Animal1].flipHorizontal = true;
00209     data[Obstacle::Type::Animal1].groupDelayTime = sf::seconds(5);
00210     data[Obstacle::Type::Animal1].groupSpawnAmount = 1;
00211     data[Obstacle::Type::Animal1].passTime =
      sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00212     data[Obstacle::Type::Animal1].noTrafficLight = true;
00213
00214     data[Obstacle::Type::Animal2].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00215     data[Obstacle::Type::Animal2].scaleX = false;
00216
00217     data[Obstacle::Type::Animal2].scaleY = false;
00218     data[Obstacle::Type::Animal2].hasAnimation = true;
00219     data[Obstacle::Type::Animal2].animation = Animations::ID::Animal2;
00220     data[Obstacle::Type::Animal2].texture = Textures::ID::Oto2;
00221     data[Obstacle::Type::Animal2].minTime = sf::seconds(1.6);
00222     data[Obstacle::Type::Animal2].maxTime = sf::seconds(2.8);
00223     data[Obstacle::Type::Animal2].flipHorizontal = true;
00224     data[Obstacle::Type::Animal2].groupDelayTime = sf::seconds(5);
00225     data[Obstacle::Type::Animal2].groupSpawnAmount = 1;
00226     data[Obstacle::Type::Animal2].passTime =
      sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00227     data[Obstacle::Type::Animal2].noTrafficLight = true;
00228
00229     data[Obstacle::Type::Animal4].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00230     data[Obstacle::Type::Animal4].scaleX = false;
00231
00232     data[Obstacle::Type::Animal4].scaleY = false;
00233     data[Obstacle::Type::Animal4].hasAnimation = true;
00234     data[Obstacle::Type::Animal4].animation = Animations::ID::Animal4;
00235     data[Obstacle::Type::Animal4].texture = Textures::ID::Oto2;
00236     data[Obstacle::Type::Animal4].minTime = sf::seconds(1.6);
00237     data[Obstacle::Type::Animal4].maxTime = sf::seconds(2.8);
00238     data[Obstacle::Type::Animal4].flipHorizontal = true;
00239     data[Obstacle::Type::Animal4].groupDelayTime = sf::seconds(5);
00240     data[Obstacle::Type::Animal4].groupSpawnAmount = 1;
00241     data[Obstacle::Type::Animal4].passTime =
      sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00242     data[Obstacle::Type::Animal4].noTrafficLight = true;
00243
00244     data[Obstacle::Type::Animal3].speed = sf::Vector2f(-Constants::carSpeed, 0.f);
00245     data[Obstacle::Type::Animal3].scaleX = false;
00246
00247     data[Obstacle::Type::Animal3].scaleY = false;
00248     data[Obstacle::Type::Animal3].hasAnimation = true;
00249     data[Obstacle::Type::Animal3].animation = Animations::ID::Animal3;
00250     data[Obstacle::Type::Animal3].texture = Textures::ID::Oto2;
00251     data[Obstacle::Type::Animal3].minTime = sf::seconds(1.6);
00252     data[Obstacle::Type::Animal3].maxTime = sf::seconds(2.8);
00253     data[Obstacle::Type::Animal3].flipHorizontal = true;
00254     data[Obstacle::Type::Animal3].groupDelayTime = sf::seconds(5);
00255     data[Obstacle::Type::Animal3].groupSpawnAmount = 1;
00256     data[Obstacle::Type::Animal3].passTime =
      sf::seconds(Constants::WindowWidth/Constants::trainSpeed);
00257     data[Obstacle::Type::Animal3].noTrafficLight = true;
00258
00259     return data;
00260
00261 }
```

6.25 Source/Entity.cpp File Reference

`#include <Entity.hpp>`
Include dependency graph for Entity.cpp:

6.26 Entity.cpp

Go to the documentation of this file.

```
00001 #include <Entity.hpp>
00002
00003 void Entity::setVelocity(sf::Vector2f velocity) {
00004     mVelocity = velocity;
00005 }
00006
00007 void Entity::setVelocity(float vx, float vy) {
00008     mVelocity.x = vx;
00009     mVelocity.y = vy;
00010 }
00011
00012 sf::Vector2f Entity::getVelocity() const {
00013     return mVelocity;
00014 }
00015
00016 void Entity::accelerate(sf::Vector2f velocity) {
00017     mVelocity += velocity;
00018 }
00019
00020 void Entity::accelerate(float vx, float vy) {
00021     mVelocity.x += vx;
00022     mVelocity.y += vy;
00023 }
00024
00025 void Entity::updateCurrent(sf::Time dt) {
00026     if (killByTime) {
00027         killTime-=dt;
00028         if (killTime.asSeconds()<0) {
00029             //kill here
00030             return;
00031         }
00032     }
00033     move(mVelocity * dt.asSeconds());
00034 }
00035
00036 void Entity::updateCurrent(sf::Time dt, CommandQueue&)
00037 {
00038     if (killByTime) {
00039         killTime-=dt;
00040         if (killTime.asSeconds()<0) {
00041             //kill here
00042             return;
00043         }
00044     }
00045     move(mVelocity * dt.asSeconds());
00046 }
```

6.27 Source/GameLevel.cpp File Reference

`#include <GameLevel.hpp>`
`#include <Const.hpp>`
Include dependency graph for GameLevel.cpp:

Variables

- GameLevel gameLevel

6.27.1 Variable Documentation

6.27.1.1 gameLevel

```
GameLevel gameLevel
```

Definition at line 95 of file GameLevel.cpp.

6.28 GameLevel.cpp

Go to the documentation of this file.

```
00001 #include <GameLevel.hpp>
00002
00003 #include <Const.hpp>
00004
00005 GameLevel::GameLevel() {
00006     load();
00007 }
00008
00009 void GameLevel::saveHighScore(TypeMap::ID typeMap) {
00010     std::vector<float> highScore = loadHighScore(typeMap);
00011     if (highScore.size() < Constants::MaxNumSaveScore) {
00012         highScore.push_back(mScore);
00013     }
00014     else {
00015         if (highScore.back() < mScore) {
00016             highScore.back() = mScore;
00017         }
00018     }
00019     sort(highScore.begin(), highScore.end(), std::greater<float>());
00020
00021     std::ofstream file;
00022     file.open(Constants::saveMapPath + "map" + std::to_string(typeMap) + ".txt");
00023     if (file.is_open()) {
00024         for (auto score : highScore) {
00025             file « score « std::endl;
00026         }
00027         file.close();
00028     }
00029 }
00030
00031 std::vector<float> GameLevel::loadHighScore(TypeMap::ID typeMap) {
00032     std::ifstream file;
00033     file.open(Constants::saveMapPath + "map" + std::to_string(typeMap) + ".txt");
00034     if (!file.is_open()) {
00035         return std::vector<float> (Constants::MaxNumSaveScore, 0);
00036     }
00037     float score;
00038     std::vector<float> highScore;
00039     while (file » score) {
00040         if (highScore.size() < Constants::MaxNumSaveScore) {
00041             highScore.push_back(score);
00042         }
00043     }
00044     file.close();
00045     while (highScore.size() < Constants::MaxNumSaveScore) {
00046         highScore.push_back(0);
00047     }
00048     return highScore;
00049 }
00050
00051 void GameLevel::load() {
00052     std::ifstream file(Constants::savePath);
00053     if (file.is_open()) {
00054         std::string line;
00055         std::getline(file, line);
00056         mScore = std::stoi(line);
00057         file.close();
00058     }
00059     else {
00060         mScore = 0;
00061     }
00062 }
00063
00064 void GameLevel::save() {
00065     std::ofstream file(Constants::savePath);
00066     if (file.is_open()) {
```

```
00067         file « mScore;
00068         file.close();
00069     }
00070 }
00071
00072 int GameLevel::getLevel() const {
00073     int level = mScore / 100;
00074     return level <= 1 ? 1 : level;
00075 }
00076
00077
00078 float GameLevel::getSpeedMultiplier() const {
00079     return (getLevel() - 1) * 0.2 + 1;
00080 }
00081
00082 void GameLevel::restart() {
00083     mScore = 0;
00084 }
00085 void GameLevel::incrementScore(float score) {
00086     mScore += score;
00087 }
00088 void GameLevel::setScore(float score) {
00089     mScore = score;
00090 }
00091 float GameLevel::getScore() const {
00092     return mScore;
00093 }
00094
00095 GameLevel gameLevel;
```

6.29 Source/GameObject.cpp File Reference

```
#include <iostream>
#include <GameObject.hpp>
```

Include dependency graph for GameObject.cpp:

6.30 GameObject.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003 #include <GameObject.hpp>
00004
00005 const std::vector<std::vector<Tile::Type» RowObject::initilizeTileTypes() {
00006     std::vector<Tile::Type> grass = {Tile::Grass};
00007     std::vector<Tile::Type> sand = {Tile::Sand};
00008     std::vector<Tile::Type> ice = {Tile::Ice};
00009     std::vector<Tile::Type> road = {Tile::Road};
00010     std::vector<Tile::Type> rail = {Tile::Rail};
00011     std::vector<Tile::Type> grass2 = {Tile::Grass};
00012     std::vector<Tile::Type> sand2 = {Tile::Sand};
00013     std::vector<Tile::Type> sand3 = {Tile::Sand};
00014     std::vector<Tile::Type> grass3 = {Tile::Grass};
00015     std::vector<Tile::Type> grass4 = {Tile::Grass};
00016     std::vector<Tile::Type> road2 = {Tile::Road};
00017     std::vector<Tile::Type> rail2 = {Tile::Rail};
00018     std::vector<Tile::Type> grass5 = {Tile::Grass};
00019     std::vector<Tile::Type> grass6 = {Tile::Grass};
00020     std::vector<Tile::Type> grass7 = {Tile::Grass};
00021     std::vector<Tile::Type> log = {Tile::Log};
00022     std::vector<Tile::Type> soil = {Tile::Soil};
00023     std::vector<std::vector<Tile::Type» tileTypes = {grass, sand, ice, road, rail, grass2, sand2,
      sand3, grass3, grass4, road2, rail2, grass5, grass6, grass7, log, soil};
00024     return tileTypes;
00025 }
00026
00027 const std::vector<std::vector<Obstacle::Type» RowObject::initilizeObstacleTypes() {
00028     std::vector<Obstacle::Type> road = {Obstacle::Car, Obstacle::Car1, Obstacle::Oto, Obstacle::Oto_1,
      Obstacle::Oto1, Obstacle::Oto1_1, Obstacle::Oto2, Obstacle::Oto2_1};
00029     std::vector<Obstacle::Type> rail = {Obstacle::Train, Obstacle::Train1};
00030     std::vector<Obstacle::Type> sand = {Obstacle::Animal3};
00031     std::vector<Obstacle::Type> grass = {Obstacle::Tree, Obstacle::Tree1, Obstacle::Tree2,
      Obstacle::Tree3, Obstacle::Tree4, Obstacle::Tree5};
00032     std::vector<Obstacle::Type> ice = {Obstacle::Island};
```

```
00033     std::vector<Obstacle::Type> grass2 = {Obstacle::Stone};
00034     std::vector<Obstacle::Type> sand2 = {};
00035     std::vector<Obstacle::Type> sand3 = {};
00036     std::vector<Obstacle::Type> grass3 = {Obstacle::Animal1, Obstacle::Animal2};
00037     std::vector<Obstacle::Type> grass4 = {};
00038     std::vector<Obstacle::Type> road2 = {Obstacle::Car, Obstacle::Car1, Obstacle::Oto,
      Obstacle::Oto_1, Obstacle::Oto1, Obstacle::Oto1_1, Obstacle::Oto2, Obstacle::Oto2_1};
00039     std::vector<Obstacle::Type> rail2 = {Obstacle::Train, Obstacle::Train1};
00040     std::vector<Obstacle::Type> grass5 = {Obstacle::SlowDown};
00041     std::vector<Obstacle::Type> grass6 = {Obstacle::SpeedUp};
00042     std::vector<Obstacle::Type> grass7 = {Obstacle::IceCream};
00043     std::vector<Obstacle::Type> log = {Obstacle::Island};
00044     std::vector<Obstacle::Type> soil = {Obstacle::Animal4};
00045     std::vector<std::vector<Obstacle::Type» obstacleTypes = {grass, sand, ice, road, rail, grass2,
      sand2, sand3, grass3, grass4, road2, rail2, grass5, grass6, grass7, log, soil};
00046     return obstacleTypes;
00047 }
00048
00049 const static std::vector<std::vector<Tile::Type» mTileTypes = RowObject::initilizeTileTypes();
00050 const static std::vector<std::vector<Obstacle::Type» mObstacleTypes =
      RowObject::initilizeObstacleTypes();
00051
00052
00053 GameObject::GameObject(sf::Vector2f spawnOrigin, std::function<sf::FloatRect()> getBattlefieldBounds,
      TextureHolder* textures, std::map<Animations::ID, Animation>& animations) :
      getBattlefieldBounds(getBattlefieldBounds), mTextures(textures), mSpawnOrigin(spawnOrigin),
      mAnimations(animations) {
00054     SceneNode::Ptr tileRow(new SceneNode);
00055     mTiles = tileRow.get();
00056     attachChild(std::move(tileRow));
00057     SceneNode::Ptr obstacleRow(new SceneNode);
00058     mObstacles = obstacleRow.get();
00059     attachChild(std::move(obstacleRow));
00060     initialGenerate();
00061 }
00062
00063 void GameObject::initialGenerate() {
00064     for (int i = 0; i < Constants::initialShift;i++) {
00065         int rowType = rand() % RowObject::TypeCount;
00066         SceneNode::Ptr tileRow(new TileRow(mTileTypes[rowType], getBattlefieldBounds, mTextures));
00067         SceneNode::Ptr obstacleRow(new ObstacleRow(mObstacleTypes[rowType], getBattlefieldBounds,
      mTextures, mAnimations));
00068         tileRow.get()->setPosition(mSpawnOrigin);
00069         obstacleRow.get()->setPosition(mSpawnOrigin);
00070         mTiles->attachChild(std::move(tileRow));
00071         mObstacles->attachChild(std::move(obstacleRow));
00072         mSpawnOrigin.y -= Constants::GridSize;
00073     }
00074     for (int i = Constants::initialShift; i <= Constants::initialShift; i++) {
00075         SceneNode::Ptr tileRow(new TileRow(mTileTypes[RowObject::Grass], getBattlefieldBounds,
      mTextures));
00076         tileRow.get()->setPosition(mSpawnOrigin);
00077         mTiles->attachChild(std::move(tileRow));
00078         mSpawnOrigin.y -= Constants::GridSize;
00079     }
00080 }
00081
00082 void GameObject::setSpawnOrigin(sf::Vector2f spawnOrigin) {
00083     mSpawnOrigin = spawnOrigin;
00084 }
00085
00086 void GameObject::updateCurrent(sf::Time dt) {
00087     sf::FloatRect battlefieldBounds = getBattlefieldBounds();
00088     while (mSpawnOrigin.y > battlefieldBounds.top) {
00089         int rowType = rand() % RowObject::TypeCount;
00090         while (rowType==last) {
00091             rowType = rand() % RowObject::TypeCount;
00092         }
00093         last=rowType;
00094         SceneNode::Ptr tileRow(new TileRow(mTileTypes[rowType], getBattlefieldBounds, mTextures));
00095         SceneNode::Ptr obstacleRow(new ObstacleRow(mObstacleTypes[rowType], getBattlefieldBounds,
      mTextures, mAnimations));
00096         tileRow.get()->setPosition(mSpawnOrigin);
00097         obstacleRow.get()->setPosition(mSpawnOrigin);
00098         mTiles->attachChild(std::move(tileRow));
00099         mObstacles->attachChild(std::move(obstacleRow));
00100         mSpawnOrigin.y -= Constants::GridSize;
00101     }
00102     removeWrecks();
00103 }
```

6.31 Source/GameOverState.cpp File Reference

```
#include <GameOverState.hpp>
#include <Utility.hpp>
#include <Player.hpp>
#include <ResourceHolder.hpp>
#include <GameLevel.hpp>
#include <MusicPlayer.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
```

Include dependency graph for GameOverState.cpp:

6.32 GameOverState.cpp

Go to the documentation of this file.

```
00001 #include <GameOverState.hpp>
00002 #include <Utility.hpp>
00003 #include <Player.hpp>
00004 #include <ResourceHolder.hpp>
00005 #include <GameLevel.hpp>
00006 #include <MusicPlayer.hpp>
00007
00008 #include <SFML/Graphics/RectangleShape.hpp>
00009 #include <SFML/Graphics/RenderWindow.hpp>
00010 #include <SFML/Graphics/View.hpp>
00011
00012
00013 GameOverState::GameOverState(StateStack& stack, Context context)
00014 : State(stack, context)
00015 , mGameOverText()
00016 , mElapsedTime(sf::Time::Zero)
00017 {
00018     sf::Font& font = context.fonts->get(Fonts::Main);
00019     sf::Vector2f windowSize(context.window->getSize());
00020
00021     mGameOverText.setFont(font);
00022     if (context.player->getMissionStatus() == Player::MissionFailure)
00023         mGameOverText.setString("GAME OVER");
00024
00025     mGameOverText.setCharacterSize(100);
00026     centerOrigin(mGameOverText);
00027     mGameOverText.setPosition(0.5f * windowSize.x, 0.4f * windowSize.y);
00028
00029     context.music->play(Music::GameOverTheme);
00030 }
00031
00032 void GameOverState::draw()
00033 {
00034     sf::RenderWindow& window = *getContext().window;
00035     window.setView(window.getDefaultView());
00036
00037     // Create dark, semitransparent background
00038     sf::RectangleShape backgroundShape;
00039     backgroundShape.setFillColor(sf::Color(0, 0, 0, 150));
00040     backgroundShape.setSize(window.getView().getSize());
00041
00042     window.draw(backgroundShape);
00043     window.draw(mGameOverText);
00044 }
00045
00046 bool GameOverState::update(sf::Time dt)
00047 {
00048     // Show state for 3 seconds, after return to menu
00049     mElapsedTime += dt;
00050     if (mElapsedTime > sf::seconds(3))
00051     {
00052         gameLevel.restart();
00053         requestStateClear();
00054         requestStackPush(States::Menu);
00055     }
00056     return false;
00057 }
00058
```

```
00059 bool GameOverState::handleEvent(const sf::Event&)
00060 {
00061     return false;
00062 }
```

6.33 Source/GameState.cpp File Reference

```
#include <GameState.hpp>
#include <GameLevel.hpp>
#include <MusicPlayer.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
```

Include dependency graph for GameState.cpp:

6.34 GameState.cpp

Go to the documentation of this file.

```
00001 #include <GameState.hpp>
00002 #include <GameLevel.hpp>
00003 #include <MusicPlayer.hpp>
00004
00005 #include <SFML/Graphics/RenderWindow.hpp>
00006
00007 GameState::GameState(StateStack& stack, Context context) : State(stack, context),
      mWorld(*context.window), mPlayer(*context.player)
00008 {
00009     mPlayer.setMissionStatus(Player::MissionRunning);
00010
00011     mLevelText.setFont(context.fonts->get(Fonts::Main));
00012     mLevelText.setPosition(5.f, 20.f);
00013     mLevelText.setCharacterSize(50);
00014     context.music->play(Music::MissionTheme);
00015 }
00016
00017 void GameState::draw() {
00018     mWorld.draw();
00019     sf::RenderWindow& mWindow = *getContext().window;
00020     mWindow.setView(mWindow.getDefaultView());
00021     mWindow.draw(mLevelText);
00022 }
00023
00024 bool GameState::update(sf::Time dt) {
00025     mWorld.update(dt);
00026
00027     if (!mWorld.hasAlivePlayer())
00028     {
00029         mPlayer.setMissionStatus(Player::MissionFailure);
00030         requestStackPush(States::GameOver);
00031         gameLevel.saveHighScore(typeOfMap);
00032     }
00033
00034     CommandQueue& commands = mWorld.getCommandQueue();
00035     mPlayer.handleRealtimeInput(commands);
00036
00037     int level = gameLevel.getScore();
00038     std::string levelString = "Score: " + std::to_string(level);
00039     mLevelText.setString(levelString);
00040
00041     return true;
00042 }
00043
00044 bool GameState::handleEvent(const sf::Event& event) {
00045     // Game input handling
00046     CommandQueue& commands = mWorld.getCommandQueue();
00047     mPlayer.handleEvent(event, commands);
00048
00049     // Escape pressed, trigger the pause screen
00050     if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape)
00051         requestStackPush(States::Pause);
00052
00053     return true;
00054 }
```

6.35 Source/HighScoreState.cpp File Reference

```
#include <HighScoreState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <iostream>
#include <MapState.hpp>
#include <GameLevel.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
```

Include dependency graph for HighScoreState.cpp:

Functions

- std::string MapID2Name (TypeMap::ID typeMap)

6.35.1 Function Documentation

6.35.1.1 MapID2Name()

```
std::string MapID2Name (
            TypeMap::ID typeMap )
```

Definition at line 114 of file HighScoreState.cpp.

6.36 HighScoreState.cpp

Go to the documentation of this file.

```
00001 #include <HighScoreState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004 #include <iostream>
00005 #include <MapState.hpp>
00006 #include <GameLevel.hpp>
00007
00008 #include <SFML/Graphics/RenderWindow.hpp>
00009
00010 HighScoreState::HighScoreState(StateStack& stack, Context context)
00011 : State(stack, context)
00012 , mGUIContainer()
00013 {
00014     loadScore();
00015     //Example
00016     /*
00017     highScore[0].first = 234;
00018     highScore[0].second = "Winter";
00019     highScore[1].first = 220;
00020     highScore[1].second = "Autumn";
00021     highScore[2].first = 123;
00022     highScore[2].second = "Spring";
00023     */
00024
00025     mBackgroundSprite.setTexture(context.textures->get(Textures::HighScore));
00026
00027     auto scoreTop1 = std::make_shared<GUI::Button>(context);
00028     scoreTop1->setPosition(650.f, 440.f);
00029     scoreTop1->setText(toString(highScore[0].first[0]), 40);
00030     scoreTop1->setCallback([this]()
00031     {
00032     });
00033
00034     auto mapTop1 = std::make_shared<GUI::Button>(context);
00035     mapTop1->setPosition(1050.f, 440.f);
00036     mapTop1->setText(highScore[0].second, 40);
```

```
00037     mapTop1->setCallback([this]()
00038     {
00039     });
00040
00041     auto scoreTop2 = std::make_shared<GUI::Button>(context);
00042     scoreTop2->setPosition(650.f, 600.f);
00043     scoreTop2->setText(toString(highScore[1].first[0]), 40);
00044     scoreTop2->setCallback([this]()
00045     {
00046     });
00047
00048     auto mapTop2 = std::make_shared<GUI::Button>(context);
00049     mapTop2->setPosition(1050.f, 600.f);
00050     mapTop2->setText(highScore[1].second, 40);
00051     mapTop2->setCallback([this]()
00052     {
00053     });
00054
00055     auto scoreTop3 = std::make_shared<GUI::Button>(context);
00056     scoreTop3->setPosition(650.f, 760.f);
00057     scoreTop3->setText(toString(highScore[2].first[0]), 40);
00058     scoreTop3->setCallback([this]()
00059     {
00060     });
00061
00062     auto mapTop3 = std::make_shared<GUI::Button>(context);
00063     mapTop3->setPosition(1050.f, 760.f);
00064     mapTop3->setText(highScore[2].second, 40);
00065     mapTop3->setCallback([this]()
00066     {
00067     });
00068
00069     auto backButton = std::make_shared<GUI::Button>(context);
00070     backButton->setPosition(70.f, 950.f);
00071     backButton->setText("Back", 40);
00072     backButton->setCallback([this]()
00073     {
00074         requestStackPop();
00075     });
00076     mGUIContainer.pack(scoreTop1);
00077     mGUIContainer.pack(mapTop1);
00078     mGUIContainer.pack(scoreTop2);
00079     mGUIContainer.pack(mapTop2);
00080     mGUIContainer.pack(scoreTop3);
00081     mGUIContainer.pack(mapTop3);
00082     mGUIContainer.pack(backButton);
00083 }
00084
00085 void HighScoreState::draw()
00086 {
00087     sf::RenderWindow& window = *getContext().window;
00088
00089     window.draw(mBackgroundSprite);
00090     window.draw(mGUIContainer);
00091 }
00092
00093 bool HighScoreState::update(sf::Time)
00094 {
00095
00096     loadScore();
00097     return false;
00098 }
00099
00100 bool HighScoreState::handleEvent(const sf::Event& event)
00101 {
00102     mGUIContainer.handleEvent(event);
00103     return false;
00104 }
00105
00106 void HighScoreState::updateLabel()
00107 {
00108 }
00109
00110 void HighScoreState::addButtonLabel(Player::Action action, float y, const std::string& text, Context
      context)
00111 {
00112 }
00113
00114 std::string MapID2Name(TypeMap::ID typeMap) {
00115     switch (typeMap) {
00116         case TypeMap::ID::Spring: return "Spring";
00117         case TypeMap::ID::Autumn: return "Autumn";
00118         case TypeMap::ID::Winter: return "Winter";
00119         case TypeMap::ID::Atlantis: return "Atlantis";
00120         case TypeMap::ID::Jura: return "Jura";
00121         default : throw "Invalid the type of map!";
00122     }
```

```
00123 }
00124
00125 void HighScoreState::loadScore() {
00126     for (int i = 0; i < 5; ++i) {
00127         highScore[i].first = GameLevel::loadHighScore(TypeMap::ID(i));
00128         highScore[i].second = MapID2Name(TypeMap::ID(i));
00129     }
00130     sort(highScore.begin(), highScore.end(), [](const std::pair<std::vector<float>, std::string>& a,
      const std::pair<std::vector<float>, std::string>& b) {
00131         return a.first[0] > b.first[0];
00132     });
00133 }
```

6.37 Source/Label.cpp File Reference

```
#include <Label.h>
#include <Utility.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for Label.cpp:

Namespaces

- namespace GUI

6.38 Label.cpp

Go to the documentation of this file.

```
00001 #include <Label.h>
00002 #include <Utility.hpp>
00003
00004 #include <SFML/Graphics/RenderStates.hpp>
00005 #include <SFML/Graphics/RenderTarget.hpp>
00006
00007 namespace GUI{
00008 Label::Label(const std::string &text, const FontHolder &fonts) : mText(text, fonts.get(Fonts::Main),
      40)
00009 {
00010 }
00011 bool Label::isSelectable() const
00012 {
00013     return false;
00014 }
00015 void Label::handleEvent(const sf::Event&){
00016
00017 }
00018 void Label::draw(sf::RenderTarget &target, sf::RenderStates states) const
00019 {
00020     states.transform *= getTransform();
00021     target.draw(mText, states);
00022 }
00023 void Label::setText(const std::string& text)
00024 {
00025     mText.setString(text);
00026 }
00027 void Label::setColor(const sf::Color& color)
00028 {
00029     mText.setFillColor(color);
00030 }
00031 }
```

6.39 Source/LoadingState.cpp File Reference

```
#include <LoadingState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
```

Include dependency graph for LoadingState.cpp:

6.40 LoadingState.cpp

Go to the documentation of this file.

```
00001 #include <LoadingState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004
00005 #include <SFML/Graphics/RenderWindow.hpp>
00006 #include <SFML/Graphics/View.hpp>
00007
00008 LoadingState::LoadingState(StateStack& stack, Context context) : State(stack, context) {
00009     sf::RenderWindow& window = *getContext().window;
00010     sf::Font& font = context.fonts->get(Fonts::Main);
00011     sf::Vector2f viewSize = window.getView().getSize();
00012
00013     mLoadingText.setFont(font);
00014     mLoadingText.setString("Loading Resources");
00015     centerOrigin(mLoadingText);
00016     mLoadingText.setPosition(viewSize.x / 2.f, viewSize.y / 2.f + 50.f);
00017
00018     mProgressBarBackground.setFillColor(sf::Color::White);
00019     mProgressBarBackground.setSize(sf::Vector2f(viewSize.x - 20, 10));
00020     mProgressBarBackground.setPosition(10, mLoadingText.getPosition().y + 40);
00021
00022     mProgressBar.setFillColor(sf::Color(100,100,100));
00023     mProgressBar.setSize(sf::Vector2f(200, 10));
00024     mProgressBar.setPosition(10, mLoadingText.getPosition().y + 40);
00025
00026     setCompletion(0.f);
00027
00028     mLoadingTask.execute();
00029 }
00030
00031 void LoadingState::draw() {
00032     sf::RenderWindow& window = *getContext().window;
00033
00034     window.setView(window.getDefaultView());
00035
00036     window.draw(mLoadingText);
00037     window.draw(mProgressBarBackground);
00038     window.draw(mProgressBar);
00039 }
00040
00041 bool LoadingState::update(sf::Time) {
00042     if (mLoadingTask.isFinished()) {
00043         requestStackPop();
00044         requestStackPush(States::Game);
00045     } else {
00046         setCompletion(mLoadingTask.getCompletion());
00047     }
00048     return true;
00049 }
00050
00051 bool LoadingState::handleEvent(const sf::Event& event) {
00052     return true;
00053 }
00054
00055 void LoadingState::setCompletion(float percent) {
00056     if (percent > 1.f) {
00057         percent = 1.f;
00058     }
00059
00060     mProgressBar.setSize(sf::Vector2f(mProgressBarBackground.getSize().x * percent,
      mProgressBar.getSize().y));
00061 }
```

6.41 Source/Main.cpp File Reference

```
#include <Application.hpp>
#include <stdexcept>
#include <iostream>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Window/Event.hpp>
#include <SFML/Graphics.hpp>
```

Include dependency graph for Main.cpp:

Functions

- int main ()

6.41.1 Function Documentation

6.41.1.1 main()

```
int main ( )
```

Definition at line 11 of file Main.cpp.

6.42 Main.cpp

Go to the documentation of this file.

```
00001 #include <Application.hpp>
00002 #include <stdexcept>
00003 #include <iostream>
00004
00005 #include <SFML/Graphics/Texture.hpp>
00006 #include <SFML/Graphics/Sprite.hpp>
00007 #include <SFML/Graphics/RenderWindow.hpp>
00008 #include <SFML/Window/Event.hpp>
00009 #include <SFML/Graphics.hpp>
00010
00011 int main()
00012 {
00013     try
00014     {
00015         Application app;
00016         app.run();
00017     }
00018     catch (std::exception& e)
00019     {
00020         std::cout << "\nEXCEPTION: " << e.what() << std::endl;
00021     }
00022     return 0;
00023 }
```

6.43 Source/MapState.cpp File Reference

```
#include <MapState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <iostream>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Image.hpp>
```

Include dependency graph for MapState.cpp:

Functions

- TypeMap::ID setTypeMap (int typeMap)

Variables

- TypeMap::ID typeOfMap

6.43.1 Function Documentation

6.43.1.1 setTypeMap()

```
TypeMap::ID setTypeMap (
             int typeMap )
```

Definition at line 9 of file MapState.cpp.

6.43.2 Variable Documentation

6.43.2.1 typeOfMap

```
TypeMap::ID typeOfMap
```

Definition at line 8 of file MapState.cpp.

6.44 MapState.cpp

Go to the documentation of this file.

```
00001 #include <MapState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004 #include <iostream>
00005
00006 #include <SFML/Graphics/RenderWindow.hpp>
00007 #include <SFML/Graphics/Image.hpp>
00008 TypeMap::ID typeOfMap;
00009 TypeMap::ID setTypeMap(int typeMap){
00010     switch (typeMap)
00011     {
00012     case 0:
00013         return TypeMap::Spring;
00014     case 1:
00015         return TypeMap::Autumn;
00016     case 2:
00017         return TypeMap::Winter;
00018     case 3:
00019         return TypeMap::Atlantis;
00020     case 4:
00021         return TypeMap::Jura;
00022     default:
00023         throw "Invalid the type of map!";
00024     }
00025 }
00026 MapState::MapState(StateStack &stack, Context context)
00027 : State(stack, context)
00028 , mGUIContainer()
00029 {
00030     mBackgroundSprite.setTexture(context.textures->get(Textures::Map));
00031     listMap[0].loadFromFile("Media/Textures/Spring.png");
00032     listMap[1].loadFromFile("Media/Textures/Autumn.png");
00033     listMap[2].loadFromFile("Media/Textures/Winter.png");
```

```
00034     listMap[3].loadFromFile("Media/Textures/Atlantis.png");
00035     listMap[4].loadFromFile("Media/Textures/Jura.png");
00036 
00037     mMap.setPosition(470, 100);
00038     mMap.setScale(0.8, 0.8);
00039     mMap.setTexture(listMap[0]);
00040 
00041     auto backButton = std::make_shared<GUI::Button>(context);
00042     backButton->setPosition(70.f, 950.f);
00043     backButton->setText("Back", 40);
00044     backButton->setCallback([this]()
00045     {
00046         requestStackPop();
00047         requestStackPush(States::Character);
00048     });
00049 
00050     auto playButton = std::make_shared<GUI::Button>(context);
00051     playButton->setPosition(1540.f, 950.f);
00052     playButton->setText("Play", 40);
00053     playButton->setCallback([this]()
00054     {
00055         requestStackPop();
00056         requestStackPush(States::Game);
00057     });
00058     mGUIContainer.pack(backButton);
00059     mGUIContainer.pack(playButton);
00060 }
00061 void MapState::draw(){
00062     sf::RenderWindow& window = *getContext().window;
00063     window.setView(window.getDefaultView());
00064 
00065     window.draw(mBackgroundSprite);
00066     window.draw(mMap);
00067     window.draw(mGUIContainer);
00068 }
00069 bool MapState::handleEvent(const sf::Event& event){
00070     if (event.type == sf::Event::KeyPressed){
00071         if (event.key.code == sf::Keyboard::D || event.key.code == sf::Keyboard::Right){
00072             if (typeMap < listMap.size() - 1){
00073                 typeMap++;
00074             }
00075         }
00076         else if (event.key.code == sf::Keyboard::A || event.key.code == sf::Keyboard::Left){
00077             if (typeMap > 0){
00078                 typeMap--;
00079             }
00080         }
00081     }
00082     mGUIContainer.handleEvent(event);
00083     return true;
00084 }
00085 bool MapState::update(sf::Time dt){
00086     mMap.setTexture(listMap[typeMap]);
00087     typeOfMap = setTypeMap(typeMap);
00088     return true;
00089 }
```

6.45 Source/MenuState.cpp File Reference

```
#include <MenuState.hpp>
#include <Utility.hpp>
#include <Foreach.hpp>
#include <ResourceHolder.hpp>
#include <Const.hpp>
#include <MovingObject.hpp>
#include <GameLevel.hpp>
#include <MusicPlayer.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
```

Include dependency graph for MenuState.cpp:

6.46 MenuState.cpp

Go to the documentation of this file.

```
00001 #include <MenuState.hpp>
00002 #include <Utility.hpp>
00003 #include <Foreach.hpp>
00004 #include <ResourceHolder.hpp>
00005 #include <Const.hpp>
00006 #include <MovingObject.hpp>
00007 #include <GameLevel.hpp>
00008 #include <MusicPlayer.hpp>
00009
00010 #include <SFML/Graphics/RenderWindow.hpp>
00011 #include <SFML/Graphics/View.hpp>
00012
00013 MenuState::MenuState(StateStack& stack, Context context) : State(stack, context), mGUIContainer() {
00014     sf::Texture& texture = context.textures->get(Textures::Background);
00015     mBackgroundSprite.setTexture(texture);
00016     mBackgroundSprite.scale(Constants::WindowWidth / mBackgroundSprite.getGlobalBounds().width,
      Constants::WindowHeight / mBackgroundSprite.getGlobalBounds().height);
00017
00018     //Animation
00019     sf::Texture& cloudTexture_1 = context.textures->get(Textures::Cloud1);
00020     sf::Texture& cloudTexture_2 = context.textures->get(Textures::Cloud2);
00021     sf::Texture& cloudTexture_3 = context.textures->get(Textures::Cloud3);
00022     sf::Texture& catTexture = context.textures->get(Textures::Cat);
00023     sf::Texture& titleTexture = context.textures->get(Textures::Title);
00024
00025
00026     std::unique_ptr<MovingObject> cloud(new MovingObject(cloudTexture_1));
00027     cloud->setPosition(250.5, 118.5);
00028     cloud->setVelocity(150, 0);
00029     clouds.attachChild(std::move(cloud));
00030
00031     std::unique_ptr<MovingObject> title(new MovingObject(titleTexture));
00032     title->setPosition(965, 239);
00033     title->setVelocity(0, 0);
00034     clouds.attachChild(std::move(title));
00035
00036     std::unique_ptr<MovingObject> cloud_1(new MovingObject(cloudTexture_2));
00037     cloud_1->setPosition(1679.5, 418);
00038     cloud_1->setVelocity(200, 0);
00039     clouds.attachChild(std::move(cloud_1));
00040
00041     std::unique_ptr<MovingObject> cloud_2(new MovingObject(cloudTexture_3));
00042     cloud_2->setPosition(128.5, 229);
00043     cloud_2->setVelocity(120, 0);
00044     clouds.attachChild(std::move(cloud_2));
00045
00046     std::unique_ptr<MovingObject> cat(new MovingObject(catTexture));
00047     cat->setPosition(57.5, 1047);
00048
00049     cat->setVelocity(100, 0);
00050     clouds.attachChild(std::move(cat));
00051
00052     // A simple menu demonstration
00053     auto playButton = std::make_shared<GUI::Button>(context);
00054     playButton->setPosition(800, 400);
00055     playButton->setText("Play", 40);
00056     playButton->setCallback([this] ()
00057     {
00058         gameLevel.restart();
00059         requestStateClear();
00060         requestStackPush(States::Character);
00061     });
00062     auto loadButton = std::make_shared<GUI::Button>(context);
00063     loadButton->setPosition(800, 500);
00064     loadButton->setText("Load Game", 40);
00065     loadButton->setCallback([this] ()
00066     {
00067         gameLevel.load();
00068         requestStateClear();
00069         requestStackPush(States::CountDown);
00070     });
00071     auto scoreButton = std::make_shared<GUI::Button>(context);
00072     scoreButton->setPosition(800, 600);
00073     scoreButton->setText("High Score", 40);
00074     scoreButton->setCallback([this]()
00075     {
00076         requestStackPush(States::Score);
00077     });
00078     auto settingButton = std::make_shared<GUI::Button>(context);
00079     settingButton->setPosition(800, 700);
00080     settingButton->setText("Setting", 40);
00081     settingButton->setCallback([this]()
00082     {
00083         requestStackPush(States::Setting);
00084     });
00085
00086     auto creditButton = std::make_shared<GUI::Button>(context);
```

```
00087     creditButton->setPosition(800, 800);
00088     creditButton->setText("Credit", 40);
00089     creditButton->setCallback([this]()
00090     {
00091         requestStackPush(States::Credit);
00092     });
00093
00094     auto exitButton = std::make_shared<GUI::Button>(context);
00095     exitButton->setPosition(800, 900);
00096     exitButton->setText("Exit", 40);
00097     exitButton->setCallback([this]()
00098     {
00099         requestStateClear();
00100         requestStackPush(States::Exit);
00101     });
00102
00103     mGUIContainer.pack(playButton);
00104     mGUIContainer.pack(loadButton);
00105     mGUIContainer.pack(scoreButton);
00106     mGUIContainer.pack(settingButton);
00107     mGUIContainer.pack(creditButton);
00108     mGUIContainer.pack(exitButton);
00109
00110     context.music->play(Music::MenuTheme);
00111 }
00112
00113 void MenuState::draw() {
00114     sf::RenderWindow& window = *getContext().window;
00115
00116     window.setView(window.getDefaultView());
00117     window.draw(mBackgroundSprite);
00118     window.draw(clouds);
00119     window.draw(mGUIContainer);
00120 }
00121
00122 bool MenuState::update(sf::Time dt) {
00123     clouds.update(dt);
00124     clouds.outOfScreen();
00125     return true;
00126 }
00127
00128 bool MenuState::handleEvent(const sf::Event& event) {
00129     mGUIContainer.handleEvent(event);
00130     return true;
00131 }
```

6.47 Source/MovingObject.cpp File Reference

`#include <MovingObject.hpp>`

Include dependency graph for MovingObject.cpp:

6.48 MovingObject.cpp

Go to the documentation of this file.

```
00001 #include <MovingObject.hpp>
00002 MovingObject::MovingObject(const sf::Texture& mTexture) : mSprite(mTexture)
00003 {
00004     mSprite.setOrigin(getGlobalBounds().width / 2, getGlobalBounds().height / 2);
00005 }
00006 MovingObject::MovingObject(const sf::Texture& mTexture, sf::IntRect bound) : mSprite(mTexture, bound)
00007 {
00008     mSprite.setOrigin(getGlobalBounds().width / 2, getGlobalBounds().height / 2);
00009 }
00010 void MovingObject::drawCurrent(sf::RenderTarget& target, sf::RenderStates states) const{
00011     target.draw(mSprite, states);
00012 }
00013 void MovingObject::setTexture(const sf::Texture& mTexture) {
00014     mSprite.setTexture(mTexture);
00015 }
00016 sf::FloatRect MovingObject::getBoundingRect() const {
00017     return getWorldTransform().transformRect(mSprite.getGlobalBounds());
00018 }
00019 sf::FloatRect MovingObject::getGlobalBounds() const {
00020     return getWorldTransform().transformRect(mSprite.getGlobalBounds());
00021 }
```

```
00022
00023 void MovingObject::flipHorizontal() {
00024     setScale(-1.0f,1.0f);
00025 }
00026
00027 void MovingObject::flipVertical() {
00028     setScale(1.0f,-1.0f);
00029 }
```

6.49 Source/MusicPlayer.cpp File Reference

```
#include <MusicPlayer.hpp>
#include <MapState.hpp>
#include <World.hpp>
```

Include dependency graph for MusicPlayer.cpp:

6.50 MusicPlayer.cpp

Go to the documentation of this file.

```
00001 #include <MusicPlayer.hpp>
00002 #include <MapState.hpp>
00003 #include <World.hpp>
00004
00005 MusicPlayer::MusicPlayer()
00006 : mMusic()
00007 , mFilenames()
00008 , mVolume(100.f)
00009 {}
00010
00011 void MusicPlayer::updateFilenames() {
00012     std::string typeMap = IDtoString(typeOfMap);
00013
00014     mFilenames[Music::MenuTheme]    = "Media/Music/MenuTheme.mp3";
00015     mFilenames[Music::MissionTheme] = "Media/Music/" + typeMap + "/MissionTheme.mp3";
00016     mFilenames[Music::GameOverTheme] = "Media/Music/GameOverTheme.mp3";
00017     mFilenames[Music::CountDownTheme] = "Media/Music/CountDownTheme.mp3";
00018 }
00019
00020 void MusicPlayer::play(Music::ID theme)
00021 {
00022     updateFilenames();
00023     std::string filename = mFilenames[theme];
00024
00025     if (!mMusic.openFromFile(filename))
00026         throw std::runtime_error("Music " + filename + " could not be loaded.");
00027
00028     mMusic.setVolume(mVolume);
00029     mMusic.setLoop(true);
00030     mMusic.play();
00031 }
00032
00033 void MusicPlayer::stop()
00034 {
00035     mMusic.stop();
00036 }
00037
00038 void MusicPlayer::setVolume(float volume)
00039 {
00040     mVolume = volume;
00041 }
00042
00043 void MusicPlayer::setPaused(bool paused)
00044 {
00045     if (paused)
00046         mMusic.pause();
00047     else
00048         mMusic.play();
00049 }
00050
00051 float MusicPlayer::getVolume()
00052 {
00053     return mVolume;
00054 }
```

6.51 Source/ObstacleManagement.cpp File Reference

```
#include <iostream>
#include <ObstacleManagement.hpp>
#include <DataTables.hpp>
#include <GameLevel.hpp>
```

Include dependency graph for ObstacleManagement.cpp:

Namespaces

- namespace ObstacleDataTables

Functions

- Textures::ID toTextureID (Obstacle::Type type)

Variables

- const std::vector< ObstacleData > ObstacleDataTables::data = initializeObstacleData()

6.51.1 Function Documentation

6.51.1.1 toTextureID()

```
Textures::ID toTextureID (
            Obstacle::Type type )
```

Definition at line 12 of file ObstacleManagement.cpp.

6.52 ObstacleManagement.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003 #include <ObstacleManagement.hpp>
00004 #include <DataTables.hpp>
00005
00006
00007 #include <GameLevel.hpp>
00008 namespace ObstacleDataTables {
00009     const std::vector<ObstacleData> data = initializeObstacleData();
00010 };
00011
00012 Textures::ID toTextureID(Obstacle::Type type) {
00013     return ObstacleDataTables::data[type].texture;
00014     switch (type) {
00015         case Obstacle::Car:
00016             return Textures::Car;
00017         case Obstacle::Oto:
00018             return Textures::Oto;
00019         case Obstacle::Train:
00020             return Textures::Train;
00021         case Obstacle::Island:
00022             return Textures::Island;
00023         case Obstacle::TrafficLightGreen:
00024             return Textures::TrafficLightGreen;
00025         case Obstacle::TrafficLightRed:
```

```
00026             return Textures::TrafficLightRed;
00027         case Obstacle::TrafficLightYellow:
00028             return Textures::TrafficLightYellow;
00029         case Obstacle::SlowDown:
00030             return Textures::SlowDown;
00031         case Obstacle::SpeedUp:
00032             return Textures::SpeedUp;
00033         case Obstacle::IceCream:
00034             return Textures::IceCream;
00035         case Obstacle::Tree:
00036             return Textures::Tree;
00037         case Obstacle::Tree1:
00038             return Textures::Tree1;
00039         case Obstacle::Tree2:
00040             return Textures::Tree2;
00041         case Obstacle::Tree3:
00042             return Textures::Tree3;
00043         case Obstacle::Tree4:
00044             return Textures::Tree4;
00045         case Obstacle::Tree5:
00046             return Textures::Tree5;
00047         default:
00048             throw std::runtime_error("Invalid obstacle type");
00049     }
00050 }
00051
00052 unsigned int Obstacle::getCategory() const {
00053     switch (mType) {
00054         case Car:
00055             return Category::Car | Category::Obstacle;
00056         case Car1:
00057             return Category::Car | Category::Obstacle;
00058         case Oto:
00059             return Category::Car | Category::Obstacle;
00060         case Oto_1:
00061             return Category::Car | Category::Obstacle;
00062         case Oto1:
00063             return Category::Car | Category::Obstacle;
00064         case Oto1_1:
00065             return Category::Car | Category::Obstacle;
00066         case Oto2:
00067             return Category::Car | Category::Obstacle;
00068         case Oto2_1:
00069             return Category::Car | Category::Obstacle;
00070         case Stone:
00071             return Category::Stone | Category::Obstacle;
00072         case Island:
00073             return Category::Island | Category::Obstacle;
00074         case Train:
00075             return Category::Car | Category::Obstacle;
00076         case Train1:
00077             return Category::Car | Category::Obstacle;
00078         case TrafficLightGreen:
00079             return Category::TrafficLightGreen;
00080         case TrafficLightRed:
00081             return Category::TrafficLightRed;
00082         case TrafficLightYellow:
00083             return Category::TrafficLightYellow;
00084         case SlowDown:
00085             return Category::SlowDown | Category::PickUp | Category::HurtLarge;
00086         case SpeedUp:
00087             return Category::SpeedUp | Category::PickUp | Category::Hot | Category::HurtSmall;
00088         case IceCream:
00089             return Category::Cold | Category::PickUp | Category::HealLarge;
00090         case Tree:
00091             return Category::Stone | Category::Obstacle;
00092         case Tree1:
00093             return Category::Stone | Category::Obstacle;
00094         case Tree2:
00095             return Category::Stone | Category::Obstacle;
00096         case Tree3:
00097             return Category::Stone | Category::Obstacle;
00098         case Tree4:
00099             return Category::Stone | Category::Obstacle;
00100         case Tree5:
00101             return Category::Stone | Category::Obstacle;
00102         case Animal1:
00103             return Category::Car | Category::HurtLarge | Category::PickUp;
00104         case Animal2:
00105             return Category::Car | Category::HurtLarge | Category::PickUp;
00106         case Animal3:
00107             return Category::Car | Category::HurtLarge | Category::PickUp;
00108         case Animal4:
00109             return Category::Car | Category::HurtLarge | Category::PickUp;
00110         default:
00111             throw std::runtime_error("Invalid obstacle type");
00112     }
```

```
00113 }
00114
00115 bool Obstacle::isDestroyed() const {
00116     if (isDestroyedFlag) return true;
00117     if (isKillByTime()) {
00118         return getKillTime().asSeconds()<0;
00119     }
00120     return !getBattlefieldBounds().intersects(getBoundingRect());
00121 }
00122
00123 sf::FloatRect Obstacle::getBoundingRect() const {
00124     if (ObstacleDataTables::data[mType].hasAnimation) {
00125         return getWorldTransform().transformRect(mAnimation.getGlobalBounds());
00126     }
00127     else {
00128         return MovingObject::getBoundingRect();
00129     }
00130 }
00131
00132
00133 Obstacle::Obstacle(Type type, const TextureHolder& textures, std::function<sf::FloatRect()>
    getBattlefieldBounds, std::map<Animations::ID, Animation>& animations) : mType(type),
    MovingObject(textures.get(toTextureID(type))), getBattlefieldBounds(getBattlefieldBounds) {
00134     if (ObstacleDataTables::data[type].hasAnimation) {
00135         mAnimation = animations[ObstacleDataTables::data[type].animation];
00136     }
00137     rotate(ObstacleDataTables::data[type].rotateAngle);
00138     if (ObstacleDataTables::data[type].flipHorizontal) flipHorizontal();
00139     if (ObstacleDataTables::data[type].flipVertical) flipVertical();
00140     if (ObstacleDataTables::data[type].scaleX) {
00141         if (ObstacleDataTables::data[type].scaleY) {
00142             scale(Constants::GridSize / getGlobalBounds().width, Constants::GridSize /
    getGlobalBounds().height);
00143         }
00144         else {
00145             scale(Constants::GridSize / getGlobalBounds().width, Constants::GridSize /
    getGlobalBounds().width);
00146         }
00147     }
00148     else {
00149         if (ObstacleDataTables::data[type].scaleY) {
00150             scale(Constants::GridSize / getGlobalBounds().height, Constants::GridSize /
    getGlobalBounds().height);
00151         }
00152         else {
00153
00154         }
00155     }
00156     if (ObstacleDataTables::data[type].killByTime) {
00157         setKillTime(ObstacleDataTables::data[type].killTime);
00158     }
00159 }
00160
00161 void Obstacle::drawCurrent(sf::RenderTarget& target, sf::RenderStates states) const {
00162     if (ObstacleDataTables::data[mType].hasAnimation) {
00163         target.draw(mAnimation, states);
00164     }
00165     else {
00166         MovingObject::drawCurrent(target, states);
00167     }
00168 }
00169
00170 void Obstacle::updateCurrent(sf::Time dt) {
00171     if (ObstacleDataTables::data[mType].hasAnimation) {
00172         mAnimation.update(dt);
00173     }
00174     else {
00175         MovingObject::updateCurrent(dt);
00176     }
00177 }
00178
00179 Obstacle::~Obstacle() {
00180 }
00181
00182 ObstacleRow::ObstacleRow(std::vector<Obstacle::Type> types, std::function<sf::FloatRect()>
    getBattlefieldBounds, TextureHolder* textures, std::map<Animations::ID, Animation>& animation) :
    getBattlefieldBounds(getBattlefieldBounds), mTextures(textures), mAnimations(animation) {
00183     if (types.empty()) {
00184         mType = Obstacle::Type::TypeCount;
00185         return;
00186     }
00187     mType = types[rand() % types.size()];
00188     groupSpawnSize = ObstacleDataTables::data[mType].groupSpawnAmount;
00189     if (groupSpawnSize==0) {
00190         groupSpawnSize=1;
00191     }
00192     groupSpawnLeft = groupSpawnSize;
```

```
00193     sf::Time minTime = ObstacleDataTables::data[mType].minTime;
00194     sf::Time maxTime = ObstacleDataTables::data[mType].maxTime;
00195     if (minTime!=maxTime) {
00196         sf::Time deltaTime = maxTime - minTime;
00197         int randTime = rand() % (int)deltaTime.asMilliseconds();
00198         sf::Time randomTime = sf::milliseconds(randTime);
00199         randomTime += minTime;
00200         randomTimeGroup = randomTime;
00201     }
00202     sf::Vector2f velocity = ObstacleDataTables::data[mType].speed;
00203     velocity *= gameLevel.getSpeedMultiplier();
00204     setVelocity(velocity);
00205     generateRow();
00206 }
00207
00208 void ObstacleRow::generateRow() {
00209     if (mType == Obstacle::Type::TypeCount) {
00210         return;
00211     }
00212     if (ObstacleDataTables::data[mType].maxDistance == 0 &&
      ObstacleDataTables::data[mType].minDistance == 0) return;
00213     int delta = rand() % ObstacleDataTables::data[mType].maxDistance;
00214     for (int i = -Constants::TilesRenderedWide; i <= Constants::TilesRenderedWide; i++) {
00215         if (delta == 0) {
00216             SceneNode::Ptr obstacle(new Obstacle(mType, *mTextures, getBattlefieldBounds,
      mAnimations));
00217             obstacle.get()->setPosition(i * Constants::GridSize, 0);
00218             attachChild(std::move(obstacle));
00219             delta = ObstacleDataTables::data[mType].minDistance + rand() %
      (ObstacleDataTables::data[mType].maxDistance - ObstacleDataTables::data[mType].minDistance);
00220         }
00221         else delta--;
00222     }
00223 }
00224
00225 sf::FloatRect ObstacleRow::getBoundingRect() const {
00226     sf::Vector2f position = getWorldPosition();
00227     position.x -= 1e9;
00228     position.y -= Constants::GridSize / 2;
00229     sf::FloatRect rect(position.x, position.y, 2e9, Constants::GridSize);
00230     return rect;
00231 }
00232
00233 bool ObstacleRow::isDestroyed() const {
00234     return !getBattlefieldBounds().intersects(getBoundingRect())|isDestroyedFlag;
00235 }
00236
00237 Obstacle::Type Obstacle::getType() {
00238     return mType;
00239 }
00240
00241 void ObstacleRow::updateCurrent(sf::Time dt) {
00242     move(getVelocity() * dt.asSeconds());
00243     for (int i=0;i<mChildren.size();i++) {
00244         if (dynamic_cast<Obstacle&>(*mChildren[i]).getType() == Obstacle::Type::TrafficLightGreen) {
00245             mChildren[i]->move(-getVelocity() * dt.asSeconds());
00246         }
00247         if (dynamic_cast<Obstacle&>(*mChildren[i]).getType() == Obstacle::Type::TrafficLightRed) {
00248             mChildren[i]->move(-getVelocity() * dt.asSeconds());
00249         }
00250     }
00251     if (mType == Obstacle::Type::TypeCount) {
00252         return;
00253     }
00254     if (mTimeToWait <= sf::Time::Zero) {
00255         if (!hasSpawned) {
00256             hasSpawned=true;
00257             //Green light spawn here
00258             if (!ObstacleDataTables::data[mType].noTrafficLight) {
00259                 SceneNode::Ptr lightObstacle(new Obstacle(Obstacle::Type::TrafficLightGreen,
      *mTextures, getBattlefieldBounds, mAnimations));
00260                 float lightLeftBound = getBattlefieldBounds().left + Constants::lightOffset -
      getPosition().x ;
00261                 float lightRightBound = getBattlefieldBounds().left + getBattlefieldBounds().width -
      Constants::lightOffset - getPosition().x ;
00262                 if (getVelocity().x > 0) {
00263                     lightObstacle->setPosition(lightLeftBound, 0);
00264                 }
00265                 else if (getVelocity().x < 0) {
00266                     lightObstacle->setPosition(lightRightBound, 0);
00267                 }
00268
      lightObstacle->setKillTime(sf::milliseconds(int(randomTimeGroup.asMilliseconds())*(groupSpawnLeft-1))+mTimeToSpawn+sf::s
00269                 attachChild(std::move(lightObstacle));
00270                 //-------------------------------------------------
00271             }
00272         }
```

```
00273         if (mTimeToSpawn <= sf::Time::Zero) {
00274             bool isSpawn = rand() % 5;
00275             if (!isSpawn) {
00276                 return;
00277             }
00278             if (getVelocity().x == 0) {
00279                 return;
00280             }
00281 
00282             SceneNode::Ptr obstacle(new Obstacle(mType, *mTextures, getBattlefieldBounds,
      mAnimations));
00283             float leftBound = getBattlefieldBounds().left +
      ObstacleDataTables::data[mType].spawnOffset - getPosition().x;
00284             float rightBound = getBattlefieldBounds().left + getBattlefieldBounds().width -
      ObstacleDataTables::data[mType].spawnOffset - getPosition().x;
00285             // obstacle->move(leftBound, 0);
00286             if (getVelocity().x > 0) {
00287                 obstacle->setPosition(leftBound, 0);
00288             }
00289             else if (getVelocity().x < 0) {
00290                 obstacle->setPosition(rightBound, 0);
00291             }
00292             attachChild(std::move(obstacle));
00293             groupSpawnLeft--;
00294             if (groupSpawnLeft==0) {
00295                 sf::Time minTime = ObstacleDataTables::data[mType].minTime;
00296                 sf::Time maxTime = ObstacleDataTables::data[mType].maxTime;
00297                 sf::Time deltaTime = maxTime - minTime;
00298                 int randTime = rand() % (int)deltaTime.asMilliseconds();
00299                 sf::Time randomTime = sf::milliseconds(randTime);
00300                 randomTime += minTime;
00301                 randomTimeGroup = randomTime;
00302                 mTimeToWait = ObstacleDataTables::data[mType].groupDelayTime;
00303                 if (!ObstacleDataTables::data[mType].noTrafficLight) {
00304                     //Spawn green light here
00305                     SceneNode::Ptr lightObstacle(new Obstacle(Obstacle::Type::TrafficLightRed,
      *mTextures, getBattlefieldBounds, mAnimations));
00306                     float lightLeftBound = getBattlefieldBounds().left - getPosition().x +
      Constants::lightOffset;
00307                     float lightRightBound = getBattlefieldBounds().left + getBattlefieldBounds().width
      - Constants::lightOffset - getPosition().x ;
00308                     if (getVelocity().x > 0) {
00309                         lightObstacle->setPosition(lightLeftBound, 0);
00310                     }
00311                     else if (getVelocity().x < 0) {
00312                         lightObstacle->setPosition(lightRightBound, 0);
00313                     }
00314 
      lightObstacle->setKillTime(randomTimeGroup+mTimeToWait+sf::seconds(Constants::trafficLightKillDelay)-ObstacleDataTables
00315                     attachChild(std::move(lightObstacle));
00316                     //-----------------------------------------
00317                 }
00318                 groupSpawnLeft = ObstacleDataTables::data[mType].groupSpawnAmount;
00319                 hasSpawned=false;
00320             }
00321             mTimeToSpawn = randomTimeGroup;
00322         }
00323         else mTimeToSpawn -= dt;
00324     }
00325     else mTimeToWait -= dt;
00326 }
```

6.53 Source/ParallelTask.cpp File Reference

`#include <ParallelTask.hpp>`

Include dependency graph for ParallelTask.cpp:

6.54 ParallelTask.cpp

Go to the documentation of this file.

```
00001 #include <ParallelTask.hpp>
00002 
00003 ParallelTask::ParallelTask() : mThread(&ParallelTask::runTask, this), mFinished(false),
      mElapsedTime(), mMutex() {}
00004 
```

```
00005 void ParallelTask::execute() {
00006     mFinished = false;
00007     mElapsedTime.restart();
00008     mThread.launch();
00009 }
00010
00011 bool ParallelTask::isFinished() {
00012     sf::Lock lock(mMutex);
00013     return mFinished;
00014 }
00015
00016 float ParallelTask::getCompletion() {
00017     sf::Lock lock(mMutex);
00018     return mElapsedTime.getElapsedTime().asSeconds() / 10.f;
00019 }
00020
00021 void ParallelTask::runTask() {
00022     bool ended = false;
00023     while (!ended) {
00024         sf::Lock lock(mMutex);
00025         if (mElapsedTime.getElapsedTime().asSeconds() >= 10.f) {
00026             ended = true;
00027         }
00028     }
00029     {
00030         sf::Lock lock(mMutex);
00031         mFinished = true;
00032     }
00033 }
```

6.55 Source/PauseState.cpp File Reference

```
#include <iostream>
#include <PauseState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <MusicPlayer.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
#include <GameLevel.hpp>
```

Include dependency graph for PauseState.cpp:

6.56 PauseState.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <PauseState.hpp>
00003 #include <Utility.hpp>
00004 #include <ResourceHolder.hpp>
00005 #include <MusicPlayer.hpp>
00006
00007 #include <SFML/Graphics/RectangleShape.hpp>
00008 #include <SFML/Graphics/RenderWindow.hpp>
00009 #include <SFML/Graphics/View.hpp>
00010
00011 #include <GameLevel.hpp>
00012
00013 PauseState::PauseState(StateStack& stack, Context context) : State(stack, context),
      mBackgroundSprite(), mPausedText(), mGUIContainer() {
00014     sf::Font& font = context.fonts->get(Fonts::Main);
00015     sf::Vector2f viewSize = context.window->getView().getSize();
00016
00017     mPausedText.setFont(font);
00018     mPausedText.setString("Game Paused");
00019     mPausedText.setCharacterSize(70);
00020     centerOrigin(mPausedText);
00021     mPausedText.setPosition(0.5f * viewSize.x, 0.2f * viewSize.y);
00022
00023     //Pause Menu
00024     auto continueButton = std::make_shared<GUI::Button>(context);
```

```
00025     continueButton->setPosition(800, 300);
00026     continueButton->setText("Continue", 30);
00027     continueButton->setCallback([this] ()
00028     {
00029         requestStackPop();
00030     });
00031     auto restartButton = std::make_shared<GUI::Button>(context);
00032     restartButton->setPosition(800, 400);
00033     restartButton->setText("Restart", 30);
00034     restartButton->setCallback([this]()
00035     {
00036         gameLevel.restart();
00037         requestStateClear();
00038         requestStackPush(States::Game);
00039     });
00040     auto settingButton = std::make_shared<GUI::Button>(context);
00041     settingButton->setPosition(800, 500);
00042     settingButton->setText("Setting", 30);
00043     settingButton->setCallback([this]()
00044     {
00045         gameLevel.save();
00046         requestStackPush(States::Setting);
00047     });
00048     auto saveAndquitButton = std::make_shared<GUI::Button>(context);
00049     saveAndquitButton->setPosition(800, 600);
00050     saveAndquitButton->setText("Save and Quit", 30);
00051     saveAndquitButton->setCallback([this]()
00052     {
00053         gameLevel.save();
00054         requestStateClear();
00055         requestStackPush(States::Saving);
00056     });
00057     auto menuButton = std::make_shared<GUI::Button>(context);
00058     menuButton->setPosition(800, 700);
00059     menuButton->setText("Return to Main Menu", 30);
00060     menuButton->setCallback([this]()
00061     {
00062         gameLevel.save();
00063         requestStateClear();
00064         requestStackPush(States::Menu);
00065     });
00066
00067     mGUIContainer.pack(continueButton);
00068     mGUIContainer.pack(restartButton);
00069     mGUIContainer.pack(settingButton);
00070     mGUIContainer.pack(saveAndquitButton);
00071     mGUIContainer.pack(menuButton);
00072
00073     getContext().music->setPaused(true);
00074 }
00075
00076 PauseState::~PauseState()
00077 {
00078     getContext().music->setPaused(false);
00079 }
00080
00081 void PauseState::draw() {
00082     sf::RenderWindow& window = *getContext().window;
00083     window.setView(window.getDefaultView());
00084
00085     sf::RectangleShape backgroundShape;
00086     backgroundShape.setFillColor(sf::Color(0, 0, 0, 150));
00087     backgroundShape.setSize(window.getView().getSize());
00088
00089     window.draw(backgroundShape);
00090     window.draw(mPausedText);
00091     window.draw(mGUIContainer);
00092 }
00093
00094 bool PauseState::update(sf::Time) {
00095     return false;
00096 }
00097
00098 bool PauseState::handleEvent(const sf::Event& event) {
00099     mGUIContainer.handleEvent(event);
00100     return false;
00101 }
```

6.57 Source/Player.cpp File Reference

```
#include <Player.hpp>
#include <CommandQueue.hpp>
```

```
#include <Character.hpp>
#include <Foreach.hpp>
#include <Const.hpp>
#include <map>
#include <string>
#include <algorithm>
```

Include dependency graph for Player.cpp:

Classes

- struct CharacterMover

6.58 Player.cpp

Go to the documentation of this file.

```
00001 #include <Player.hpp>
00002 #include <CommandQueue.hpp>
00003 #include <Character.hpp>
00004 #include <Foreach.hpp>
00005 #include <Const.hpp>
00006
00007 #include <map>
00008 #include <string>
00009 #include <algorithm>
00010
00011 struct CharacterMover {
00012     CharacterMover(float vx, float vy) : velocity(vx, vy) {}
00013
00014     void operator() (Character& character, sf::Time) const {
00015         character.pathRequest(velocity);
00016     }
00017
00018     sf::Vector2f velocity;
00019 };
00020
00021 Player::Player() {
00022     // Set initial key bindings
00023     mKeyBinding[sf::Keyboard::Left] = MoveLeft;
00024     mKeyBinding[sf::Keyboard::Right] = MoveRight;
00025     mKeyBinding[sf::Keyboard::Up] = MoveUp;
00026     mKeyBinding[sf::Keyboard::Down] = MoveDown;
00027
00028     // Set initial action bindings
00029     initializeActions();
00030
00031     // Assign all categories to player's character
00032     FOREACH(auto& pair, mActionBinding)
00033         pair.second.category = Category::PlayerCharacter;
00034 }
00035
00036 void Player::handleEvent(const sf::Event& event, CommandQueue& commands) {
00037     if (event.type == sf::Event::KeyPressed) {
00038         // Check if pressed key appears in key binding, trigger command if so
00039         auto found = mKeyBinding.find(event.key.code);
00040         if (found != mKeyBinding.end() && !isRealtimeAction(found->second))
00041             commands.push(mActionBinding[found->second]);
00042     }
00043 }
00044
00045 void Player::handleRealtimeInput(CommandQueue& commands) {
00046     // Traverse all assigned keys and check if they are pressed
00047     FOREACH(auto pair, mKeyBinding) {
00048         // If key is pressed, lookup action and trigger corresponding command
00049         if (sf::Keyboard::isKeyPressed(pair.first) && isRealtimeAction(pair.second))
00050             commands.push(mActionBinding[pair.second]);
00051     }
00052 }
00053
00054 void Player::assignKey(Action action, sf::Keyboard::Key key) {
00055     // Remove all keys that already map to action
00056     for (auto itr = mKeyBinding.begin(); itr != mKeyBinding.end(); ) {
00057         if (itr->second == action)
00058             mKeyBinding.erase(itr++);
00059         else
```

```
00060             ++itr;
00061     }
00062
00063     // Insert new binding
00064     mKeyBinding[key] = action;
00065 }
00066
00067 sf::Keyboard::Key Player::getAssignedKey(Action action) const {
00068     FOREACH(auto pair, mKeyBinding) {
00069         if (pair.second == action)
00070             return pair.first;
00071     }
00072
00073     return sf::Keyboard::Unknown;
00074 }
00075
00076 void Player::setMissionStatus(MissionStatus status)
00077 {
00078     mCurrentMissionStatus = status;
00079 }
00080
00081 Player::MissionStatus Player::getMissionStatus() const
00082 {
00083     return mCurrentMissionStatus;
00084 }
00085
00086 void Player::initializeActions() {
00087     const float playerSpeed = Constants::playerSpeed;
00088
00089     mActionBinding[MoveLeft].action = derivedAction<Character>(CharacterMover(-playerSpeed, 0.f));
00090     mActionBinding[MoveRight].action = derivedAction<Character>(CharacterMover(+playerSpeed, 0.f));
00091     mActionBinding[MoveUp].action = derivedAction<Character>(CharacterMover(0.f, -playerSpeed));
00092     mActionBinding[MoveDown].action = derivedAction<Character>(CharacterMover(0.f, +playerSpeed));
00093 }
00094
00095 bool Player::isRealtimeAction(Action action) {
00096     switch (action) {
00097         case MoveLeft:
00098         case MoveRight:
00099         case MoveDown:
00100         case MoveUp:
00101             return false;
00102
00103         default:
00104             return false;
00105     }
00106 }
```

6.59 Source/SavingState.cpp File Reference

```
#include <SavingState.hpp>
#include <Utility.hpp>
#include <Player.hpp>
#include <ResourceHolder.hpp>
#include <GameLevel.hpp>
#include <MapState.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/View.hpp>
```

Include dependency graph for SavingState.cpp:

6.60 SavingState.cpp

Go to the documentation of this file.

```
00001 #include <SavingState.hpp>
00002 #include <Utility.hpp>
00003 #include <Player.hpp>
00004 #include <ResourceHolder.hpp>
00005 #include <GameLevel.hpp>
00006 #include <MapState.hpp>
00007
```

```
00008 #include <SFML/Graphics/RectangleShape.hpp>
00009 #include <SFML/Graphics/RenderWindow.hpp>
00010 #include <SFML/Graphics/View.hpp>
00011
00012
00013 SavingState::SavingState(StateStack& stack, Context context)
00014 : State(stack, context)
00015 , mSavingText()
00016 , mElapsedTime(sf::Time::Zero)
00017 {
00018     sf::Font& font = context.fonts->get(Fonts::Main);
00019     sf::Vector2f windowSize(context.window->getSize());
00020
00021     mSavingText.setFont(font);
00022     mSavingText.setString("SAVING... \nPLEASE DON'T BREAK ANYTHING");
00023     mSavingText.setCharacterSize(100);
00024     centerOrigin(mSavingText);
00025     mSavingText.setPosition(0.5f * windowSize.x, 0.4f * windowSize.y);
00026 }
00027
00028 void SavingState::draw()
00029 {
00030     sf::RenderWindow& window = *getContext().window;
00031     window.setView(window.getDefaultView());
00032
00033     // Create dark, semitransparent background
00034     sf::RectangleShape backgroundShape;
00035     backgroundShape.setFillColor(sf::Color(0, 0, 0, 150));
00036     backgroundShape.setSize(window.getView().getSize());
00037
00038     window.draw(backgroundShape);
00039     window.draw(mSavingText);
00040 }
00041
00042 bool SavingState::update(sf::Time dt)
00043 {
00044     // Show state for 3 seconds, after return to menu
00045     gameLevel.save();
00046     mElapsedTime += dt;
00047     if (mElapsedTime > sf::seconds(3))
00048     {
00049         gameLevel.restart();
00050         requestStateClear();
00051         requestStackPush(States::Title);
00052     }
00053     return false;
00054 }
00055
00056 bool SavingState::handleEvent(const sf::Event&)
00057 {
00058     return false;
00059 }
```

6.61 Source/SceneNode.cpp File Reference

```
#include <iostream>
#include <Command.hpp>
#include <SceneNode.hpp>
#include <Foreach.hpp>
#include <Const.hpp>
#include <Utility.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <algorithm>
#include <cassert>
#include <cmath>
```

Include dependency graph for SceneNode.cpp:

6.62 SceneNode.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003 #include <Command.hpp>
00004 #include <SceneNode.hpp>
00005 #include <Foreach.hpp>
00006 #include <Const.hpp>
00007 #include <Utility.hpp>
00008
00009 #include <SFML/Graphics/RectangleShape.hpp>
00010 #include <SFML/Graphics/RenderTarget.hpp>
00011
00012 #include <algorithm>
00013 #include <cassert>
00014 #include <cmath>
00015 using namespace std;
00016 SceneNode::SceneNode(Category::Type category) : mChildren(), mParent(nullptr),
      mDefaultCategory(category) {}
00017
00018 void SceneNode::attachChild(Ptr child) {
00019     child->mParent = this;
00020     mChildren.push_back(std::move(child));
00021 }
00022
00023 SceneNode::Ptr SceneNode::detachChild(const SceneNode& node) {
00024     auto found = std::find_if(mChildren.begin(), mChildren.end(), [&] (Ptr& p) {
00025         return p.get() == &node;
00026     });
00027     assert(found != mChildren.end());
00028
00029     Ptr result = std::move(*found);
00030     result->mParent = nullptr;
00031     mChildren.erase(found);
00032     return result;
00033 }
00034
00035 void SceneNode::update(sf::Time dt) {
00036     updateCurrent(dt);
00037     updateChildren(dt);
00038 }
00039
00040 void SceneNode::updateCurrent(sf::Time) {
00041     // Do nothing by default
00042 }
00043
00044 void SceneNode::updateChildren(sf::Time dt) {
00045     FOREACH(Ptr& child, mChildren)
00046         child->update(dt);
00047 }
00048
00049 void SceneNode::draw(sf::RenderTarget& target, sf::RenderStates states) const {
00050     if (isHide()) return;
00051     states.transform *= getTransform();
00052
00053     drawCurrent(target, states);
00054     drawChildren(target, states);
00055
00056     // drawBoundingRect(target, states);
00057 }
00058
00059 void SceneNode::drawCurrent(sf::RenderTarget&, sf::RenderStates) const {
00060     // Do nothing by default
00061 }
00062
00063 void SceneNode::drawChildren(sf::RenderTarget& target, sf::RenderStates states) const {
00064     FOREACH(const Ptr& child, mChildren)
00065         child->draw(target, states);
00066 }
00067
00068 sf::Vector2f SceneNode::getWorldPosition() const {
00069     return getWorldTransform() * sf::Vector2f();
00070 }
00071
00072 sf::Transform SceneNode::getWorldTransform() const {
00073     sf::Transform transform = sf::Transform::Identity;
00074
00075     for (const SceneNode* node = this; node != nullptr; node = node->mParent)
00076         transform = node->getTransform() * transform;
00077
00078     return transform;
00079 }
00080
00081 void SceneNode::onCommand(const Command& command, sf::Time dt) {
00082     if (command.category & getCategory())
00083         command.action(*this, dt);
00084
00085     FOREACH(Ptr& child, mChildren)
00086         child->onCommand(command, dt);
```

```
00087 }
00088
00089 unsigned int SceneNode::getCategory() const {
00090     return mDefaultCategory;
00091 }
00092
00093 void SceneNode::outOfScreen(){
00094     FOREACH(const Ptr& child, mChildren){
00095         if (child->getPosition().x > Constants::WindowWidth){
00096             float height = child->getPosition().y;
00097             child->setPosition(sf::Vector2f(-100, height));
00098         }
00099     }
00100 }
00101
00102 sf::FloatRect SceneNode::getBoundingRect() const {
00103     return sf::FloatRect();
00104 }
00105
00106 bool SceneNode::collision(const sf::FloatRect& rect) const {
00107     return getBoundingRect().intersects(rect);
00108 }
00109
00110 void SceneNode::drawBoundingRect(sf::RenderTarget& target, sf::RenderStates) const {
00111     sf::FloatRect rect = getBoundingRect();
00112
00113     sf::RectangleShape shape;
00114     shape.setPosition(sf::Vector2f(rect.left, rect.top));
00115     shape.setSize(sf::Vector2f(rect.width, rect.height));
00116     shape.setFillColor(sf::Color::Transparent);
00117     shape.setOutlineColor(sf::Color::Green);
00118     shape.setOutlineThickness(1.f);
00119
00120     target.draw(shape);
00121 }
00122
00123 void SceneNode::checkNodeCollision(const sf::FloatRect& rect, std::set<SceneNode*>& collisionNodes) {
00124     if (collision(rect))
00125         collisionNodes.insert(this);
00126     FOREACH(Ptr& child, mChildren)
00127         child->checkNodeCollision(rect, collisionNodes);
00128 }
00129
00130 bool SceneNode::isDestroyed() const {
00131     if (isDestroyedFlag) return true;
00132     return false;
00133 }
00134
00135 void SceneNode::setDestroy() {
00136     isDestroyedFlag=true;
00137 }
00138
00139 void SceneNode::removeWrecks() {
00140     auto wreckfieldBegin = std::remove_if(mChildren.begin(), mChildren.end(),
      std::mem_fn(&SceneNode::isDestroyed));
00141     mChildren.erase(wreckfieldBegin, mChildren.end());
00142
00143     std::for_each(mChildren.begin(), mChildren.end(), std::mem_fn(&SceneNode::removeWrecks));
00144 }
00145
00146 void SceneNode::hide() {
00147     isHideFlag = true;
00148 }
00149
00150 void SceneNode::show() {
00151     isHideFlag = false;
00152 }
00153
00154 bool SceneNode::isHide() const {
00155     return isHideFlag;
00156 }
00157
00158 SceneNode* SceneNode::getParent() {
00159     return mParent;
00160 }
00161
00162 void SceneNode::setKillTime(sf::Time dt) {
00163     killByTime=true;
00164     killTime=dt;
00165 }
00166
00167 bool SceneNode::isKillByTime() const {
00168     return killByTime;
00169 }
00170
00171 sf::Time SceneNode::getKillTime() const {
00172     return killTime;
```

```
00173 }
```

6.63 Source/SettingState.cpp File Reference

```
#include <iostream>
#include <SettingState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
```

Include dependency graph for SettingState.cpp:

6.64 SettingState.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <SettingState.hpp>
00003 #include <Utility.hpp>
00004 #include <ResourceHolder.hpp>
00005
00006 #include <SFML/Graphics/RenderWindow.hpp>
00007
00008 SettingState::SettingState(StateStack& stack, Context context)
00009 : State(stack, context)
00010 , mSoundButton(context)
00011 , mGUIContainer()
00012 {
00013     mBackgroundSprite.setTexture(context.textures->get(Textures::Background));
00014     mVolume.setFont(context.fonts->get(Fonts::Main));
00015     mVolume.setCharacterSize(40);
00016     mVolume.setFillColor(sf::Color::Black);
00017
00018     float weight = 768*(1 - 0.1*volume);
00019     mSound1.setTexture(context.textures->get(Textures::Sound1));
00020     mSound1.setPosition(500, 620);
00021
00022     mSound2.setTexture(context.textures->get(Textures::Sound2));
00023     mSound2.setScale(1 - 0.1* volume, 1);
00024     mSound2.setPosition(500 + 0.1*volume*768, 620);
00025
00026
00027     circle.setFillColor(sf::Color::Black);
00028     circle.setPosition(1245, 615);
00029     circle.setRadius(23);
00030
00031     addButtonLabel(Player::MoveLeft,  200.f, "Move Left", context);
00032     addButtonLabel(Player::MoveRight, 300.f, "Move Right", context);
00033     addButtonLabel(Player::MoveUp,    400.f, "Move Up", context);
00034     addButtonLabel(Player::MoveDown,  500.f, "Move Down", context);
00035     addButtonLabel(Player::Sound, 600.0f, "Sound", context);
00036
00037     updateLabel();
00038
00039     mSoundButton.setPosition(80.f, 600.f);
00040     mSoundButton.setText("Sound", 30);
00041
00042     auto backButton = std::make_shared<GUI::Button>(context);
00043     backButton->setPosition(80.f, 700.f);
00044     backButton->setText("Back", 30);
00045     backButton->setCallback([this]()
00046     {
00047         requestStackPop();
00048     });
00049     mGUIContainer.pack(backButton);
00050 }
00051
00052 void SettingState::draw()
00053 {
00054     sf::RenderWindow& window = *getContext().window;
00055     window.setView(window.getDefaultView());
00056
00057     window.draw(mBackgroundSprite);
00058     if (isSettingSound == true){
00059         window.draw(mSound1);
```

```
00060         window.draw(mSound2);
00061
00062         window.draw(circle);
00063     }
00064     window.draw(mVolume);
00065     window.draw(mGUIContainer);
00066 }
00067
00068 bool SettingState::update(sf::Time)
00069 {
00070     return false;
00071 }
00072
00073 bool SettingState::handleEvent(const sf::Event& event)
00074 {
00075     bool isKeyBinding = false;
00076
00077     for (std::size_t action = 0; action < Player::ActionCount - 1; ++action)
00078     {
00079         if (mBindingButtons[action]->isActive())
00080         {
00081             isKeyBinding = true;
00082             if (event.type == sf::Event::KeyPressed)
00083             {
00084                 getContext().player->assignKey(static_cast<Player::Action>(action), event.key.code);
00085                 mBindingButtons[action]->deactivate();
00086             }
00087             break;
00088         }
00089     }
00090     if (mBindingButtons[Player::Sound]->isActive()){
00091         isKeyBinding = true;
00092         isSettingSound = true;
00093         mVolume.setPosition(500 + 76.8*volume - 23, 550);
00094         mVolume.setString(toString(10*volume) + "%");
00095         MusicPlayer& myMusic = *getContext().music;
00096         if (event.type == sf::Event::KeyPressed){
00097             if (event.key.code == sf::Keyboard::Right)
00098             {
00099                 if (volume < 10)
00100                 {
00101                     volume++;
00102                     float newVol = myMusic.getVolume() + 4.f;
00103                     myMusic.setVolume(newVol);
00104                 }
00105             }
00106             else if (event.key.code == sf::Keyboard::Left)
00107             {
00108                 if (volume >= 1)
00109                 {
00110                     volume--;
00111                     float newVol = myMusic.getVolume() - 4.f;
00112                     myMusic.setVolume(newVol);
00113                 }
00114             }
00115             else if (event.key.code == sf::Keyboard::Return){
00116                 mBindingButtons[Player::Sound]->deactivate();
00117                 mVolume.setString(toString(10*volume) + "%");
00118                 mVolume.setPosition(500.f, 615.f);
00119                 isSettingSound = false;
00120                 return false;
00121             }
00122             mVolume.setPosition(500 + 76.8*volume - 23, 550);
00123             mVolume.setString(toString(10*volume) + "%");
00124
00125             circle.setPosition(500 + 76.8*volume - 23, 615);
00126             mSound2.setScale(1 - 0.1* volume, 1);
00127             mSound2.setPosition(500 + 0.1*volume*768, 620);
00128         }
00129     }
00130     if (isKeyBinding)
00131         updateLabel();
00132     else
00133     {
00134         mGUIContainer.handleEvent(event);
00135     }
00136     return false;
00137 }
00138
00139 void SettingState::updateLabel()
00140 {
00141     Player& player = *getContext().player;
00142
00143     for (std::size_t i = 0; i < Player::ActionCount - 1; ++i)
00144     {
00145         sf::Keyboard::Key key = player.getAssignedKey(static_cast<Player::Action>(i));
00146         mBindingLabels[i]->setText(fromKtoS(key));
```

```
00147     }
00148 }
00149
00150 void SettingState::addButtonLabel(Player::Action action, float y, const std::string& text, Context
      context)
00151 {
00152     mBindingButtons[action] = std::make_shared<GUI::Button>(context);
00153     mBindingButtons[action]->setPosition(80.f, y);
00154     mBindingButtons[action]->setText(text, 30);
00155     mBindingButtons[action]->setToggle(true);
00156
00157     mBindingLabels[action] = std::make_shared<GUI::Label>("", *context.fonts);
00158     mBindingLabels[action]->setColor(sf::Color::Black);
00159     mBindingLabels[action]->setPosition(500.f, y + 15.f);
00160
00161     mGUIContainer.pack(mBindingButtons[action]);
00162     mGUIContainer.pack(mBindingLabels[action]);
00163 }
00164 std::string SettingState::fromKtoS(const sf::Keyboard::Key& key){
00165     std::string ret;
00166     switch(key){
00167     case sf::Keyboard::A :
00168
00169         ret="A";
00170         break;
00171     case sf::Keyboard::B :
00172
00173         ret="B";
00174         break;
00175     case sf::Keyboard::C :
00176
00177         ret="C";
00178         break;
00179     case sf::Keyboard::D :
00180
00181         ret="D";
00182         break;
00183     case sf::Keyboard::E :
00184
00185         ret="E";
00186         break;
00187     case sf::Keyboard::F :
00188
00189         ret="F";
00190         break;
00191     case sf::Keyboard::G :
00192
00193         ret="G";
00194         break;
00195     case sf::Keyboard::H :
00196
00197         ret="H";
00198         break;
00199     case sf::Keyboard::I :
00200
00201         ret="I";
00202         break;
00203     case sf::Keyboard::J :
00204
00205         ret="J";
00206         break;
00207     case sf::Keyboard::K :
00208
00209         ret="K";
00210         break;
00211     case sf::Keyboard::L :
00212
00213         ret="L";
00214         break;
00215     case sf::Keyboard::M :
00216
00217         ret="M";
00218         break;
00219     case sf::Keyboard::N :
00220
00221         ret="N";
00222         break;
00223     case sf::Keyboard::O :
00224
00225         ret="O";
00226         break;
00227     case sf::Keyboard::P :
00228
00229         ret="P";
00230         break;
00231     case sf::Keyboard::Q :
00232
```

```
00233         ret="Q";
00234         break;
00235     case sf::Keyboard::R :
00236 
00237         ret="R";
00238         break;
00239     case sf::Keyboard::S :
00240 
00241         ret="S";
00242         break;
00243     case sf::Keyboard::T :
00244 
00245         ret="T";
00246         break;
00247     case sf::Keyboard::U :
00248 
00249         ret="U";
00250         break;
00251     case sf::Keyboard::V :
00252 
00253         ret="V";
00254         break;
00255     case sf::Keyboard::W :
00256 
00257         ret="W";
00258         break;
00259     case sf::Keyboard::X :
00260 
00261         ret="X";
00262         break;
00263     case sf::Keyboard::Y :
00264 
00265         ret="Y";
00266         break;
00267     case sf::Keyboard::Z :
00268 
00269         ret="Z";
00270         break;
00271     case sf::Keyboard::Num0 :
00272 
00273         ret="Num0";
00274         break;
00275     case sf::Keyboard::Num1 :
00276 
00277         ret="Num1";
00278         break;
00279     case sf::Keyboard::Num2 :
00280 
00281         ret="Num2";
00282         break;
00283     case sf::Keyboard::Num3 :
00284 
00285         ret="Num3";
00286         break;
00287     case sf::Keyboard::Num4 :
00288 
00289         ret="Num4";
00290         break;
00291     case sf::Keyboard::Num5 :
00292 
00293         ret="Num5";
00294         break;
00295     case sf::Keyboard::Num6 :
00296 
00297         ret="Num6";
00298         break;
00299     case sf::Keyboard::Num7 :
00300 
00301         ret="Num7";
00302         break;
00303     case sf::Keyboard::Num8 :
00304 
00305         ret="Num8";
00306         break;
00307     case sf::Keyboard::Num9 :
00308 
00309         ret="Num9";
00310         break;
00311     case sf::Keyboard::Escape :
00312 
00313         ret="Escape";
00314         break;
00315     case sf::Keyboard::LControl :
00316 
00317         ret="LControl";
00318         break;
00319     case sf::Keyboard::LShift :
```

```
00320
00321         ret="LShift";
00322         break;
00323     case sf::Keyboard::LAlt :
00324
00325         ret="LAlt";
00326         break;
00327     case sf::Keyboard::LSystem :
00328
00329         ret="LSystem";
00330         break;
00331     case sf::Keyboard::RControl :
00332
00333         ret="RControl";
00334         break;
00335     case sf::Keyboard::RShift :
00336
00337         ret="RShift";
00338         break;
00339     case sf::Keyboard::RAlt :
00340
00341         ret="RAlt";
00342         break;
00343     case sf::Keyboard::RSystem :
00344
00345         ret="RSystem";
00346         break;
00347     case sf::Keyboard::Menu :
00348
00349         ret="Menu";
00350         break;
00351     case sf::Keyboard::LBracket :
00352
00353         ret="LBracket";
00354         break;
00355     case sf::Keyboard::RBracket :
00356
00357         ret="RBracket";
00358         break;
00359     case sf::Keyboard::SemiColon :
00360
00361         ret="SemiColon";
00362         break;
00363     case sf::Keyboard::Comma :
00364
00365         ret="Comma";
00366         break;
00367     case sf::Keyboard::Period :
00368
00369         ret="Period";
00370         break;
00371     case sf::Keyboard::Quote :
00372
00373         ret="Quote";
00374         break;
00375     case sf::Keyboard::Slash :
00376
00377         ret="Slash";
00378         break;
00379     case sf::Keyboard::BackSlash :
00380
00381         ret="BackSlash";
00382         break;
00383     case sf::Keyboard::Tilde :
00384
00385         ret="Tilde";
00386         break;
00387     case sf::Keyboard::Equal :
00388
00389         ret="Equal";
00390         break;
00391     case sf::Keyboard::Dash :
00392
00393         ret="Dash";
00394         break;
00395     case sf::Keyboard::Space :
00396
00397         ret="Space";
00398         break;
00399     case sf::Keyboard::Return :
00400
00401         ret="Return";
00402         break;
00403     case sf::Keyboard::BackSpace :
00404
00405         ret="BackSpace";
00406         break;
```

```
00407     case sf::Keyboard::Tab :
00408
00409         ret="Tab";
00410         break;
00411     case sf::Keyboard::PageUp :
00412
00413         ret="PageUp";
00414         break;
00415     case sf::Keyboard::PageDown :
00416
00417         ret="PageDown";
00418         break;
00419     case sf::Keyboard::End :
00420
00421         ret="End";
00422         break;
00423     case sf::Keyboard::Home :
00424
00425         ret="Home";
00426         break;
00427     case sf::Keyboard::Insert :
00428
00429         ret="Insert";
00430         break;
00431     case sf::Keyboard::Delete :
00432
00433         ret="Delete";
00434         break;
00435     case sf::Keyboard::Add :
00436
00437         ret="Add";
00438         break;
00439     case sf::Keyboard::Subtract :
00440
00441         ret="Subtract";
00442         break;
00443     case sf::Keyboard::Multiply :
00444
00445         ret="Multiply";
00446         break;
00447     case sf::Keyboard::Divide :
00448
00449         ret="Divide";
00450         break;
00451     case sf::Keyboard::Left :
00452
00453         ret="Left";
00454         break;
00455     case sf::Keyboard::Right :
00456
00457         ret="Right";
00458         break;
00459     case sf::Keyboard::Up :
00460
00461         ret="Up";
00462         break;
00463     case sf::Keyboard::Down :
00464
00465         ret="Down";
00466         break;
00467     case sf::Keyboard::Numpad0 :
00468
00469         ret="Numpad0";
00470         break;
00471     case sf::Keyboard::Numpad1 :
00472
00473         ret="Numpad1";
00474         break;
00475     case sf::Keyboard::Numpad2 :
00476
00477         ret="Numpad2";
00478         break;
00479     case sf::Keyboard::Numpad3 :
00480
00481         ret="Numpad3";
00482         break;
00483     case sf::Keyboard::Numpad4 :
00484
00485         ret="Numpad4";
00486         break;
00487     case sf::Keyboard::Numpad5 :
00488
00489         ret="Numpad5";
00490         break;
00491     case sf::Keyboard::Numpad6 :
00492
00493         ret="Numpad6";
```

```
00494         break;
00495     case sf::Keyboard::Numpad7 :
00496
00497         ret="Numpad7";
00498         break;
00499     case sf::Keyboard::Numpad8 :
00500
00501         ret="Numpad8";
00502         break;
00503     case sf::Keyboard::Numpad9 :
00504
00505         ret="Numpad9";
00506         break;
00507     case sf::Keyboard::F1 :
00508
00509         ret="F1";
00510         break;
00511     case sf::Keyboard::F2 :
00512
00513         ret="F2";
00514         break;
00515     case sf::Keyboard::F3 :
00516
00517         ret="F3";
00518         break;
00519     case sf::Keyboard::F4 :
00520
00521         ret="F4";
00522         break;
00523     case sf::Keyboard::F5 :
00524
00525         ret="F5";
00526         break;
00527     case sf::Keyboard::F6 :
00528
00529         ret="F6";
00530         break;
00531     case sf::Keyboard::F7 :
00532
00533         ret="F7";
00534         break;
00535     case sf::Keyboard::F8 :
00536
00537         ret="F8";
00538         break;
00539     case sf::Keyboard::F9 :
00540
00541         ret="F9";
00542         break;
00543     case sf::Keyboard::F10 :
00544
00545         ret="F10";
00546         break;
00547     case sf::Keyboard::F11 :
00548
00549         ret="F11";
00550         break;
00551     case sf::Keyboard::F12 :
00552
00553         ret="F12";
00554         break;
00555     case sf::Keyboard::F13 :
00556
00557         ret="F13";
00558         break;
00559     case sf::Keyboard::F14 :
00560
00561         ret="F14";
00562         break;
00563     case sf::Keyboard::F15 :
00564
00565         ret="F15";
00566         break;
00567     case sf::Keyboard::Pause :
00568
00569         ret="Pause";
00570         break;
00571     case sf::Keyboard::KeyCount :
00572
00573         ret="KeyCount";
00574         break;
00575     default:
00576         ret="Unknow";
00577         break;
00578         }
00579     return ret;
00580 }
```

6.65 Source/SoundPlayer.cpp File Reference

```
#include <SoundPlayer.hpp>
#include <SFML/Audio/Listener.hpp>
#include <cmath>
```

Include dependency graph for SoundPlayer.cpp:

6.66 SoundPlayer.cpp

Go to the documentation of this file.

```
00001 #include <SoundPlayer.hpp>
00002
00003 #include <SFML/Audio/Listener.hpp>
00004
00005 #include <cmath>
00006
00007
00008 namespace
00009 {
00010     // Sound coordinate system, point of view of a player in front of the screen:
00011     // X = left; Y = up; Z = back (out of the screen)
00012     const float ListenerZ = 300.f;
00013     const float Attenuation = 8.f;
00014     const float MinDistance2D = 200.f;
00015     const float MinDistance3D = std::sqrt(MinDistance2D*MinDistance2D + ListenerZ*ListenerZ);
00016 }
00017
00018 SoundPlayer::SoundPlayer()
00019 : mSoundBuffers()
00020 , mSounds()
00021 {
00022     mSoundBuffers.load(SoundEffects::Button, "Media/Sound/Button.mp3");
00023     mSoundBuffers.load(SoundEffects::Hurt, "Media/Sound/Hurt.mp3");
00024     mSoundBuffers.load(SoundEffects::Heal, "Media/Sound/Heal.mp3");
00025     mSoundBuffers.load(SoundEffects::Die, "Media/Sound/Death.mp3");
00026     // Listener points towards the screen (default in SFML)
00027     sf::Listener::setDirection(0.f, 0.f, -1.f);
00028 }
00029
00030 void SoundPlayer::play(SoundEffects::ID effect)
00031 {
00032     play(effect, getListenerPosition());
00033 }
00034
00035 void SoundPlayer::play(SoundEffects::ID effect, sf::Vector2f position)
00036 {
00037     mSounds.push_back(sf::Sound());
00038     sf::Sound& sound = mSounds.back();
00039
00040     sound.setBuffer(mSoundBuffers.get(effect));
00041     sound.setPosition(position.x, -position.y, 0.f);
00042     sound.setAttenuation(Attenuation);
00043     sound.setMinDistance(MinDistance3D);
00044
00045     sound.play();
00046 }
00047
00048 void SoundPlayer::removeStoppedSounds()
00049 {
00050     mSounds.remove_if([] (const sf::Sound& s)
00051     {
00052         return s.getStatus() == sf::Sound::Stopped;
00053     });
00054 }
00055
00056 void SoundPlayer::setListenerPosition(sf::Vector2f position)
00057 {
00058     sf::Listener::setPosition(position.x, -position.y, ListenerZ);
00059 }
00060
00061 sf::Vector2f SoundPlayer::getListenerPosition() const
00062 {
00063     sf::Vector3f position = sf::Listener::getPosition();
00064     return sf::Vector2f(position.x, -position.y);
00065 }
```

6.67 Source/SpriteNode.cpp File Reference

```
#include <SpriteNode.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for SpriteNode.cpp:

6.68 SpriteNode.cpp

Go to the documentation of this file.

```
00001 #include <SpriteNode.hpp>
00002
00003 #include <SFML/Graphics/RenderTarget.hpp>
00004
00005 SpriteNode::SpriteNode(const sf::Texture& texture) : mSprite(texture) {}
00006
00007 SpriteNode::SpriteNode(const sf::Texture& texture, const sf::IntRect& textureRect) : mSprite(texture,
      textureRect) {}
00008
00009 void SpriteNode::drawCurrent(sf::RenderTarget& target, sf::RenderStates states) const {
00010     target.draw(mSprite, states);
00011 }
```

6.69 Source/State.cpp File Reference

```
#include <State.hpp>
#include <StateStack.hpp>
```

Include dependency graph for State.cpp:

6.70 State.cpp

Go to the documentation of this file.

```
00001 #include <State.hpp>
00002 #include <StateStack.hpp>
00003
00004 State::Context::Context(sf::RenderWindow& window, TextureHolder& textures, FontHolder& fonts, Player&
      player, MusicPlayer& music, SoundPlayer& sounds) : window(&window), textures(&textures),
      fonts(&fonts), player(&player), music(&music), sounds(&sounds) {}
00005
00006 State::State(StateStack& stack, Context context) : mStack(&stack), mContext(context) {}
00007
00008 State::~State() {}
00009
00010 void State::requestStackPush(States::ID stateID) {
00011     mStack->pushState(stateID);
00012 }
00013
00014 void State::requestStackPop() {
00015     mStack->popState();
00016 }
00017
00018 void State::requestStateClear() {
00019     mStack->clearStates();
00020 }
00021
00022 State::Context State::getContext() const {
00023     return mContext;
00024 }
```

6.71 Source/StateStack.cpp File Reference

```
#include <iostream>
#include <StateStack.hpp>
#include <Foreach.hpp>
#include <cassert>
```

Include dependency graph for StateStack.cpp:

6.72 StateStack.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <StateStack.hpp>
00003 #include <Foreach.hpp>
00004
00005 #include <cassert>
00006
00007
00008 StateStack::StateStack(State::Context context) : mStack(), mPendingList(), mContext(context),
      mFactories() {}
00009
00010 void StateStack::update(sf::Time dt) {
00011     // Iterate from top to bottom, stop as soon as update() returns false
00012     for (auto itr = mStack.rbegin(); itr != mStack.rend(); ++itr) {
00013         if (!(*itr)->update(dt)) {
00014             break;
00015         }
00016     }
00017
00018     applyPendingChanges();
00019 }
00020
00021 void StateStack::draw() {
00022     // Draw all active states from bottom to top
00023     FOREACH(State::Ptr& state, mStack) {
00024         state->draw();
00025     }
00026 }
00027
00028 void StateStack::handleEvent(const sf::Event& event) {
00029     // Iterate from top to bottom, stop as soon as handleEvent() returns false
00030     // std::cout << "mStack size: " << mStack.size() << "\n";
00031     for (auto itr = mStack.rbegin(); itr != mStack.rend(); ++itr) {
00032         if (!(*itr)->handleEvent(event)) {
00033             break;
00034         }
00035     }
00036
00037     applyPendingChanges();
00038 }
00039
00040 void StateStack::pushState(States::ID stateID) {
00041     mPendingList.push_back(PendingChange(Push, stateID));
00042 }
00043
00044 void StateStack::popState() {
00045     mPendingList.push_back(PendingChange(Pop));
00046 }
00047
00048 void StateStack::clearStates() {
00049     mPendingList.push_back(PendingChange(Clear));
00050 }
00051
00052 bool StateStack::isEmpty() const {
00053     return mStack.empty();
00054 }
00055
00056 State::Ptr StateStack::createState(States::ID stateID) {
00057     auto found = mFactories.find(stateID);
00058     assert(found != mFactories.end());
00059
00060     return found->second();
00061 }
00062
00063 void StateStack::applyPendingChanges() {
00064     FOREACH(PendingChange change, mPendingList) {
00065         switch (change.action) {
```

```
00066             case Push:
00067                 mStack.push_back(createState(change.stateID));
00068                 break;
00069 
00070             case Pop:
00071                 mStack.pop_back();
00072                 break;
00073 
00074             case Clear:
00075                 mStack.clear();
00076                 break;
00077         }
00078     }
00079 
00080     mPendingList.clear();
00081 }
00082 
00083 StateStack::PendingChange::PendingChange(Action action, States::ID stateID) : action(action),
      stateID(stateID) {}
```

6.73 Source/TextureHolder.cpp File Reference

6.74 TextureHolder.cpp

Go to the documentation of this file.

6.75 Source/TileManagement.cpp File Reference

```
#include <iostream>
#include <TileManagement.hpp>
```

Include dependency graph for TileManagement.cpp:

Functions

- Textures::ID toTextureID (Tile::Type type)

6.75.1 Function Documentation

6.75.1.1 toTextureID()

```
Textures::ID toTextureID (
            Tile::Type type )
```

Definition at line 6 of file TileManagement.cpp.

6.76 TileManagement.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003
00004 #include <TileManagement.hpp>
00005
00006 Textures::ID toTextureID(Tile::Type type) {
00007     switch (type) {
00008         case Tile::Grass:
00009             return Textures::Grass;
00010         case Tile::Sand:
00011             return Textures::Sand;
00012         case Tile::Ice:
00013             return Textures::Ice;
00014         case Tile::Log:
00015             return Textures::Log;
00016         case Tile::Road:
00017             return Textures::Road;
00018         case Tile::Rail:
00019             return Textures::Rail;
00020         case Tile::Soil:
00021             return Textures::Soil;
00022         default:
00023             throw std::runtime_error("Invalid tile type");
00024     }
00025 }
00026 unsigned int Tile::getCategory() const {
00027     switch (mType) {
00028         case Grass:
00029             return Category::Grass | Category::Tile;
00030         case Sand:
00031             return Category::Sand | Category::Tile;
00032         case Ice:
00033             return Category::Ice | Category::Tile;
00034         case Log:
00035             return Category::Ice | Category::Tile;
00036         case Road:
00037             return Category::Road | Category::Tile;
00038         case Rail:
00039             return Category::Rail | Category::Tile;
00040         case Soil:
00041             return Category::Road | Category::Tile;
00042         default:
00043             throw std::runtime_error("Invalid tile type");
00044     }
00045 }
00046 void Tile::destroy() {
00047     isDestroy = true;
00048 }
00049 bool Tile::isDestroyed() const {
00050     return isDestroy|isDestroyedFlag;
00051 }
00052 Tile::Tile(Type type, const TextureHolder& textures) : mType(type),
      MovingObject(textures.get(toTextureID(type))) {
00053     scale(Constants::GridSize / getGlobalBounds().width, Constants::GridSize /
      getGlobalBounds().height);
00054 }
00055 Tile::~Tile() {}
00056
00057
00058
00059 TileRow::TileRow(std::vector<Tile::Type> types, std::function<sf::FloatRect()> getBattlefieldBounds,
      TextureHolder* textures) : getBattlefieldBounds(getBattlefieldBounds), mTextures(textures) {
00060     generateRow(types);
00061 }
00062 void TileRow::generateRow(std::vector<Tile::Type> types) {
00063     for (int i = -Constants::TilesRenderedWide; i <= Constants::TilesRenderedWide; i++) {
00064         int type = rand() % types.size();
00065         SceneNode::Ptr tile(new Tile(types[type], *mTextures));
00066         tile.get()->setPosition(i * Constants::GridSize, 0);
00067         attachChild(std::move(tile));
00068     }
00069 }
00070
00071 sf::FloatRect TileRow::getBoundingRect() const {
00072     sf::Vector2f position = getWorldPosition();
00073     position.x -= Constants::GridSize * Constants::TilesRenderedWide;
00074     position.y -= Constants::GridSize / 2;
00075     sf::FloatRect rect(position.x, position.y, Constants::GridSize * Constants::TilesRenderedWide * 2,
      Constants::GridSize);
00076     return rect;
00077 }
00078
```

```
00079 bool TileRow::isDestroyed() const {
00080     return !getBattlefieldBounds().intersects(getBoundingRect())|isDestroyedFlag;
00081 }
00082
00083
00084 TileManager::TileManager(std::function<sf::FloatRect()> getBattlefieldBounds, TextureHolder* textures)
      : mSpawnOrigin(0, 0), getBattlefieldBounds(getBattlefieldBounds), mTextures(textures) {
00085 }
00086
00087
00088 TileManager::TileManager(sf::Vector2f spawnOrigin, std::function<sf::FloatRect()>
      getBattlefieldBounds, TextureHolder* textures) : mSpawnOrigin(spawnOrigin),
      getBattlefieldBounds(getBattlefieldBounds), mTextures(textures) {
00089 }
00090
00091 void TileManager::updateCurrent(sf::Time dt) {
00092 }
00093
00094 void TileManager::setSpawnOrigin(sf::Vector2f spawnOrigin) {
00095     mSpawnOrigin = spawnOrigin;
00096 }
```

6.77 Source/TitleState.cpp File Reference

```
#include <TitleState.hpp>
#include <Utility.hpp>
#include <ResourceHolder.hpp>
#include <Const.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
```

Include dependency graph for TitleState.cpp:

6.78 TitleState.cpp

Go to the documentation of this file.

```
00001 #include <TitleState.hpp>
00002 #include <Utility.hpp>
00003 #include <ResourceHolder.hpp>
00004 #include <Const.hpp>
00005 #include <SFML/Graphics/RenderWindow.hpp>
00006
00007 TitleState::TitleState(StateStack& stack, Context context) : State(stack, context), mText(),
      mShowText(true), mTextEffectTime(sf::Time::Zero) {
00008     mBackgroundSprite.setTexture(context.textures->get(Textures::TitleScreen));
00009     mBackgroundSprite.scale(Constants::WindowWidth / mBackgroundSprite.getGlobalBounds().width,
      Constants::WindowHeight / mBackgroundSprite.getGlobalBounds().height);
00010     mText.setFont(context.fonts->get(Fonts::Main));
00011     mText.setString("Press any key to start");
00012     centerOrigin(mText);
00013     mText.setPosition(context.window->getView().getSize() / 2.f);
00014 }
00015
00016 void TitleState::draw() {
00017     sf::RenderWindow& window = *getContext().window;
00018     window.draw(mBackgroundSprite);
00019
00020     if (mShowText)
00021         window.draw(mText);
00022 }
00023
00024 bool TitleState::update(sf::Time dt) {
00025     mTextEffectTime += dt;
00026
00027     if (mTextEffectTime >= sf::seconds(0.5f)) {
00028         mShowText = !mShowText;
00029         mTextEffectTime = sf::Time::Zero;
00030     }
00031
00032     return true;
00033 }
00034
00035 bool TitleState::handleEvent(const sf::Event& event) {
00036     // If any key is pressed, trigger the next screen
00037     if (event.type == sf::Event::KeyPressed) {
```

```
00038         requestStackPop();
00039         requestStackPush(States::Menu);
00040     }
00041
00042     return true;
00043 }
```

6.79 Source/Utility.cpp File Reference

```
#include <Utility.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/Text.hpp>
#include <cmath>
```

Include dependency graph for Utility.cpp:

Functions

- void centerOrigin (sf::Sprite &sprite)
- void centerOrigin (sf::Text &text)
- float Rand (float l, float r)
- int Rand (int l, int r)
- float toDegree (float radian)
- float toRadian (float degree)

6.79.1 Function Documentation

6.79.1.1 centerOrigin() [1/2]

```
void centerOrigin (
            sf::Sprite & sprite )
```

Definition at line 8 of file Utility.cpp.

6.79.1.2 centerOrigin() [2/2]

```
void centerOrigin (
            sf::Text & text )
```

Definition at line 13 of file Utility.cpp.

6.79.1.3 Rand() [1/2]

```
float Rand (
            float l,
            float r )
```

Definition at line 18 of file Utility.cpp.

Here is the call graph for this function: Here is the caller graph for this function:

6.79.1.4 Rand() [2/2]

```
int Rand (
            int l,
            int r )
```

Definition at line 22 of file Utility.cpp.

6.79.1.5 toDegree()

```
float toDegree (
            float radian )
```

Definition at line 26 of file Utility.cpp.

6.79.1.6 toRadian()

```
float toRadian (
            float degree )
```

Definition at line 31 of file Utility.cpp.

6.80 Utility.cpp

Go to the documentation of this file.

```
00001 #include <Utility.hpp>
00002
00003 #include <SFML/Graphics/Sprite.hpp>
00004 #include <SFML/Graphics/Text.hpp>
00005
00006 #include <cmath>
00007
00008 void centerOrigin(sf::Sprite& sprite) {
00009     sf::FloatRect bounds = sprite.getLocalBounds();
00010     sprite.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::floor(bounds.top +
      bounds.height / 2.f));
00011 }
00012
00013 void centerOrigin(sf::Text& text) {
00014     sf::FloatRect bounds = text.getLocalBounds();
00015     text.setOrigin(std::floor(bounds.left + bounds.width / 2.f), std::floor(bounds.top + bounds.height
      / 2.f));
00016 }
00017
00018 float Rand(float l,float r) {
00019     return float(Rand(int(l*1000),int(r*1000)))/1000;
00020 }
00021
00022 int Rand(int l,int r) {
00023     return (rand()%(r-l+1))+l;
00024 }
00025
00026 float toDegree(float radian)
00027 {
00028     return 180.f / 3.14159265358979323846264338f * radian;
00029 }
00030
00031 float toRadian(float degree)
00032 {
00033     return 3.14159265358979323846264338f / 180.f * degree;
00034 }
00035
```

6.81 Source/World.cpp File Reference

```
#include <World.hpp>
#include <Utility.hpp>
#include <MapState.hpp>
#include <WeatherState.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Image.hpp>
#include <algorithm>
#include <cmath>
#include <fstream>
#include <iostream>
#include <vector>
#include <GameLevel.hpp>
```

Include dependency graph for World.cpp:

Functions

- std::vector< std::pair< int, sf::Vector2i > > setAnimation (TypeMap::ID typeOfMap)
- std::string IDtoString (TypeMap::ID typeOfMap)
- bool matchesCategories (SceneNode *node, Category::Type type)

6.81.1 Function Documentation

6.81.1.1 IDtoString()

```
std::string IDtoString (
            TypeMap::ID typeOfMap )
```

Definition at line 60 of file World.cpp.

6.81.1.2 matchesCategories()

```
bool matchesCategories (
            SceneNode * node,
            Category::Type type )
```

Definition at line 431 of file World.cpp.

6.81.1.3 setAnimation()

```
std::vector< std::pair< int, sf::Vector2i > > setAnimation (
            TypeMap::ID typeOfMap )
```

Definition at line 16 of file World.cpp.

6.82 World.cpp

Go to the documentation of this file.

```
00001 #include <World.hpp>
00002 #include <Utility.hpp>
00003 #include <MapState.hpp>
00004 #include <WeatherState.hpp>
00005
00006 #include <SFML/Graphics/RenderWindow.hpp>
00007 #include <SFML/Graphics/Image.hpp>
00008
00009 #include <algorithm>
00010 #include <cmath>
00011 #include <fstream>
00012 #include <iostream>
00013 #include <vector>
00014 #include <GameLevel.hpp>
00015
00016 std::vector<std::pair<int, sf::Vector2i» setAnimation(TypeMap::ID typeOfMap){
00017     std::pair<int, sf::Vector2i> animal1;
00018     std::pair<int, sf::Vector2i> animal2;
00019     std::pair<int, sf::Vector2i> animal3;
00020     std::pair<int, sf::Vector2i> animal4;
00021     switch (typeOfMap){
00022         case TypeMap::Spring:
00023             animal1 = std::make_pair(4, sf::Vector2i(16, 16));
00024             animal2 = std::make_pair(4, sf::Vector2i(16, 16));
00025             animal3 = std::make_pair(4, sf::Vector2i(16, 16));
00026             animal4 = std::make_pair(4, sf::Vector2i(16, 16));
00027             return {animal1, animal2, animal3, animal4};
00028         case TypeMap::Autumn:
00029             animal1 = std::make_pair(4, sf::Vector2i(16, 16));
00030             animal2 = std::make_pair(4, sf::Vector2i(16, 16));
00031             animal3 = std::make_pair(4, sf::Vector2i(16, 16));
00032             animal4 = std::make_pair(4, sf::Vector2i(16, 16));
00033             return {animal1, animal2, animal3, animal4};
00034         case TypeMap::Winter:
00035             animal1 = std::make_pair(4, sf::Vector2i(16, 16));
00036             animal2 = std::make_pair(4, sf::Vector2i(16, 16));
00037             animal3 = std::make_pair(4, sf::Vector2i(16, 16));
00038             animal4 = std::make_pair(4, sf::Vector2i(16, 16));
00039             return {animal1, animal2, animal3, animal4};
00040         case TypeMap::Atlantis:
00041             animal1 = std::make_pair(8, sf::Vector2i(48, 32));
00042             animal2 = std::make_pair(4, sf::Vector2i(32, 16));
00043             animal3 = std::make_pair(8, sf::Vector2i(48, 32));
00044             animal4 = std::make_pair(8, sf::Vector2i(32, 32));
00045             return {animal1, animal2, animal3, animal4};
00046         case TypeMap::Jura:
00047             animal1 = std::make_pair(4, sf::Vector2i(50, 32));
00048             animal2 = std::make_pair(4, sf::Vector2i(50, 40));
00049             animal3 = std::make_pair(4, sf::Vector2i(50, 40));
00050             animal4 = std::make_pair(4, sf::Vector2i(16, 16));
00051             return {animal1, animal2, animal3, animal4};
00052         default:
00053             animal1 = std::make_pair(4, sf::Vector2i(16, 16));
00054             animal2 = std::make_pair(4, sf::Vector2i(16, 16));
00055             animal3 = std::make_pair(4, sf::Vector2i(16, 16));
00056             animal4 = std::make_pair(4, sf::Vector2i(16, 16));
00057             return {animal1, animal2, animal3, animal4};
00058     }
00059 }
00060 std::string IDtoString(TypeMap::ID typeOfMap){
00061     switch (typeOfMap){
00062     case TypeMap::Spring:
00063         return "Spring";
00064     case TypeMap::Autumn:
00065         return "Autumn";
00066     case TypeMap::Winter:
00067         return "Winter";
00068     case TypeMap::Atlantis:
00069         return "Atlantis";
00070     case TypeMap::Jura:
00071         return "Jura";
00072     default:
00073         return "Spring";
00074     }
00075 }
00076
00077 std::string World::getMap() {
00078     switch (typeOfMap){
00079     case TypeMap::Spring:
00080         return "Spring";
00081     case TypeMap::Autumn:
00082         return "Autumn";
```

```
00083     case TypeMap::Winter:
00084         return "Winter";
00085     case TypeMap::Atlantis:
00086         return "Atlantis";
00087     case TypeMap::Jura:
00088         return "Jura";
00089     default:
00090         return "Spring";
00091     }
00092 }
00093
00094 World::World(sf::RenderWindow& window) : lastWeatherState(0), mWindow(window),
      mWorldView(window.getDefaultView()), mTextures(), mSceneGraph(), mSceneLayers(), mWorldBounds(0.f,
      0.f, /*mWorldView.getSize().x*/ 200000.f, 200000.f), mSpawnPosition(mWorldView.getSize().x / 2.f,
      mWorldBounds.height - mWorldView.getSize().y / 2.f), mScrollSpeed(Constants::scrollSpeed),
      mPlayerCharacter(nullptr) {
00095     loadTextures();
00096     loadAnimations();
00097     buildScene();
00098
00099     // Prepare the view
00100     mWorldView.setCenter(mSpawnPosition);
00101 }
00102
00103 void World::update(sf::Time dt) {
00104
00105     if (typeOfMap==TypeMap::Spring) {
00106         clearWeather();
00107         mPlayerCharacter->setDefaultTemperature(Constants::defaultTemperatureSpring);
00108     }
00109     else if (typeOfMap==TypeMap::Autumn) {
00110         setWeather(Weather::Rain);
00111         mPlayerCharacter->setDefaultTemperature(Constants::defaultTemperatureAutumn);
00112     }
00113     else if (typeOfMap==TypeMap::Winter) {
00114         setWeather(Weather::Snowing);
00115         mPlayerCharacter->setDefaultTemperature(Constants::defaultTemperatureWinter);
00116     }
00117     else {
00118         clearWeather();
00119         mPlayerCharacter->setDefaultTemperature(Constants::defaultTemperatureSpring);
00120     }
00121
00122
00123     if (!mPlayerCharacter->isDestroyed()) mWorldView.move(0.f, mScrollSpeed * dt.asSeconds() *
      gameLevel.getSpeedMultiplier());
00124     mPlayerCharacter->setVelocity(0.f, 0.f);
00125
00126     // Forward commands to scene graph, adapt velocity (scrolling, diagonal correction)
00127     while (!mCommandQueue.isEmpty()) {
00128         mSceneGraph.onCommand(mCommandQueue.pop(), dt);
00129     }
00130     adaptPlayerVelocity();
00131
00132     handleCollisions();
00133
00134
00135     // Regular update step, adapt position (correct if outside view)
00136     mSceneGraph.update(dt);
00137     adaptPlayerPosition();
00138
00139     // Update level
00140     gameLevel.incrementScore(dt.asSeconds() * Constants::ScorePerSecond);
00141
00142     if (mPlayerCharacter->isBurning()) {
00143         if (!checkCState(CState::Type::burning)) {
00144             charState|=CState::Type::burning;
00145             screenEffect.setTexture(mTextures.get(Textures::ID::Burning));
00146             screenEffect.setNumFrames(20);
00147             screenEffect.setDuration(sf::seconds(0.5f));
00148             screenEffect.setRepeating(true);
00149             screenEffect.setFrameSize(sf::Vector2i(200,108));
00150             screenEffect.setPosition(0,0);
00151             screenEffect.restart();
00152             screenEffect.setScale(1,1);
00153
      screenEffect.setScale((Constants::WindowWidth)/(screenEffect.getGlobalBounds().width),(Constants::WindowHeight)/(screenE
00154         }
00155     }
00156     else {
00157         charState&=~CState::Type::burning;
00158     }
00159
00160     if (mPlayerCharacter->isFreezing()) {
00161         if (!checkCState(CState::Type::freezing)) {
00162             charState|=CState::Type::freezing;
00163             screenEffect.setTexture(mTextures.get(Textures::ID::Freezing));
```

```
00164             screenEffect.setNumFrames(20);
00165             screenEffect.setDuration(sf::seconds(3.f));
00166             screenEffect.setFrameSize(sf::Vector2i(320,180));
00167             screenEffect.setPosition(0,0);
00168             screenEffect.restart();
00169             screenEffect.setScale(1,1);
00170
      screenEffect.setScale((Constants::WindowWidth)/(screenEffect.getGlobalBounds().width),(Constants::WindowHeight)/(screenE
00171         }
00172     }
00173     else {
00174         charState&=~CState::Type::freezing;
00175     }
00176
00177     if (!charState) {
00178         screenEffect.hide();
00179     }
00180     else {
00181         screenEffect.show();
00182     }
00183
00184     if (isWeather(Weather::Rain)) {
00185         if (!checkLastWeatherState(Weather::Rain)) {
00186             lastWeatherState=Weather::Rain;
00187             weatherEffect.setTexture(mTextures.get(Textures::ID::Raining));
00188             weatherEffect.setNumFrames(20);
00189             weatherEffect.setDuration(sf::seconds(1.f));
00190             weatherEffect.setRepeating(true);
00191             weatherEffect.setFrameSize(sf::Vector2i(320,180));
00192             weatherEffect.setPosition(0,0);
00193             weatherEffect.restart();
00194             weatherEffect.setScale(1,1);
00195
      weatherEffect.setScale((Constants::WindowWidth)/(weatherEffect.getGlobalBounds().width),(Constants::WindowHeight)/(weath
00196         }
00197     }
00198     else if (isWeather(Weather::Snowing)) {
00199         if (!checkLastWeatherState(Weather::Snowing)) {
00200             lastWeatherState=Weather::Snowing;
00201             weatherEffect.setTexture(mTextures.get(Textures::ID::Snowing));
00202             weatherEffect.setNumFrames(20);
00203             weatherEffect.setDuration(sf::seconds(10.f));
00204             weatherEffect.setFrameSize(sf::Vector2i(320,180));
00205             weatherEffect.setPosition(0,0);
00206             weatherEffect.restart();
00207             weatherEffect.setScale(1,1);
00208
      weatherEffect.setScale((Constants::WindowWidth)/(weatherEffect.getGlobalBounds().width),(Constants::WindowHeight)/(weath
00209         }
00210     }
00211
00212     if (!weatherState) {
00213         weatherEffect.hide();
00214     }
00215     else {
00216         weatherEffect.show();
00217     }
00218
00219     if (screenEffect.isBuilt()) screenEffect.update(dt);
00220     if (weatherEffect.isBuilt()) weatherEffect.update(dt);
00221
00222     if (!heartEffect.isBuilt()) {
00223         heartEffect.setTexture(mTextures.get(Textures::ID::Heart));
00224         heartEffect.setNumFrames(20);
00225         heartEffect.setDuration(sf::seconds(1.f));
00226         heartEffect.setRepeating(true);
00227         heartEffect.setFrameSize(sf::Vector2i(320,180));
00228         heartEffect.setPosition(Constants::WindowWidth-300,100);
00229         heartEffect.restart();
00230         heartEffect.setScale(1,1);
00231     }
00232
00233     if (heartEffect.isBuilt()) heartEffect.update(dt);
00234
00235     if (mPlayerCharacter->getHealth()<=0) {
00236         mPlayerCharacter->destroy();
00237     }
00238 }
00239
00240 void World::draw() {
00241     mWindow.setView(mWorldView);
00242     mWindow.draw(mSceneGraph);
00243     mWindow.setView(mWindow.getDefaultView());
00244     mWindow.draw(weatherEffect);
00245     mWindow.setView(mWindow.getDefaultView());
00246     mWindow.draw(screenEffect);
00247     mWindow.setView(mWindow.getDefaultView());
```

```
00248     if (mPlayerCharacter->getHealth()>0) {
00249         heartEffect.setPosition(Constants::WindowWidth-200,-50);
00250         mWindow.draw(heartEffect);
00251     }
00252     if (mPlayerCharacter->getHealth()>100) {
00253         heartEffect.setPosition(Constants::WindowWidth-300,-50);
00254         mWindow.draw(heartEffect);
00255     }
00256     if (mPlayerCharacter->getHealth()>200) {
00257         heartEffect.setPosition(Constants::WindowWidth-400,-50);
00258         mWindow.draw(heartEffect);
00259     }
00260     if (mPlayerCharacter->getHealth()>300) {
00261         heartEffect.setPosition(Constants::WindowWidth-500,-50);
00262         mWindow.draw(heartEffect);
00263     }
00264     if (mPlayerCharacter->getHealth()>400) {
00265         heartEffect.setPosition(Constants::WindowWidth-600,-50);
00266         mWindow.draw(heartEffect);
00267     }
00268 }
00269
00270 CommandQueue& World::getCommandQueue() {
00271     return mCommandQueue;
00272 }
00273
00274 bool World::hasAlivePlayer() const
00275 {
00276     return !mPlayerCharacter->isMarkedForRemoval();
00277 }
00278
00279 void World::loadTextures() {
00280     std::string typeMap = IDtoString(typeOfMap);
00281
00282     mTextures.load(Textures::Player, "Media/Textures/Eagle.png");//Base Player
00283
00284     mTextures.load(Textures::Background, "Media/Textures/Desert.png");
00285
00286     mTextures.load(Textures::Grass, "Media/Textures/" + typeMap + "/Tile/Tile1.png");
00287     mTextures.load(Textures::Sand, "Media/Textures/" + typeMap + "/Tile/Tile2.png");
00288     mTextures.load(Textures::Ice, "Media/Textures/" + typeMap +  "/Vehicle/Raft.png");
00289     mTextures.load(Textures::Car, "Media/Textures/" + typeMap + "/Vehicle/Truck.png");
00290     mTextures.load(Textures::Oto, "Media/Textures/" + typeMap + "/Vehicle/Oto.png");
00291     mTextures.load(Textures::Oto1, "Media/Textures/" + typeMap + "/Vehicle/Oto1.png");
00292     mTextures.load(Textures::Oto2, "Media/Textures/" + typeMap +  "/Vehicle/Oto2.png");
00293     mTextures.load(Textures::Road, "Media/Textures/" + typeMap +  "/Tile/Tile4.png");
00294     mTextures.load(Textures::Soil, "Media/Textures/" + typeMap + "/Tile/Tile6.png");
00295     mTextures.load(Textures::Rail, "Media/Textures/" + typeMap + "/Tile/Rail.png");
00296     mTextures.load(Textures::Train, "Media/Textures/" + typeMap + "/Vehicle/Train.png");
00297     mTextures.load(Textures::Island, "Media/Textures/" + typeMap + "/Tile/Tile5.png");
00298     mTextures.load(Textures::Stone, "Media/Textures/" + typeMap + "/Vehicle/Stone.png");
00299     mTextures.load(Textures::Log, "Media/Textures/" + typeMap + "/Vehicle/Raft1.png");
00300
00301
00302     mTextures.load(Textures::Tree, "Media/Textures/" + typeMap + "/Tree/tree.png");
00303     mTextures.load(Textures::Tree1, "Media/Textures/" + typeMap + "/Tree/tree1.png");
00304     mTextures.load(Textures::Tree2, "Media/Textures/" + typeMap + "/Tree/tree2.png");
00305     mTextures.load(Textures::Tree3, "Media/Textures/" + typeMap + "/Tree/tree3.png");
00306     mTextures.load(Textures::Tree4, "Media/Textures/" + typeMap + "/Tree/tree4.png");
00307     mTextures.load(Textures::Tree5, "Media/Textures/" + typeMap + "/Tree/tree5.png");
00308
00309     mTextures.load(Textures::TrafficLightGreen, "Media/Textures/" + typeMap +
      "/TrafficLightGreen.png");
00310     mTextures.load(Textures::TrafficLightRed, "Media/Textures/" + typeMap + "/TrafficLightRed.png");
00311     mTextures.load(Textures::TrafficLightYellow, "Media/Textures/" + typeMap +
      "/TrafficLightYellow.png");
00312
00313
00314     mTextures.load(Textures::BlueDino, "Media/Textures/Characters/Moving/BlueDino.png");
00315     mTextures.load(Textures::RedDino, "Media/Textures/Characters/Moving/RedDino.png");
00316     mTextures.load(Textures::GreenDino, "Media/Textures/Characters/Moving/GreenDino.png");
00317     mTextures.load(Textures::YellowDino, "Media/Textures/Characters/Moving/YellowDino.png");
00318
00319     mTextures.load(Textures::BlueFrog, "Media/Textures/Characters/Moving/BlueFrog.png");
00320     mTextures.load(Textures::GreenFrog, "Media/Textures/Characters/Moving/GreenFrog.png");
00321     mTextures.load(Textures::PinkFrog, "Media/Textures/Characters/Moving/PinkFrog.png");
00322     mTextures.load(Textures::YellowFrog, "Media/Textures/Characters/Moving/YellowFrog.png");
00323
00324     mTextures.load(Textures::BlueDinoDeath, "Media/Textures/Characters/Death/BlueDino.png");
00325     mTextures.load(Textures::RedDinoDeath, "Media/Textures/Characters/Death/RedDino.png");
00326     mTextures.load(Textures::GreenDinoDeath, "Media/Textures/Characters/Death/GreenDino.png");
00327     mTextures.load(Textures::YellowDinoDeath, "Media/Textures/Characters/Death/YellowDino.png");
00328
00329     mTextures.load(Textures::BlueFrogDeath, "Media/Textures/Characters/Death/BlueFrog.png");
00330     mTextures.load(Textures::GreenFrogDeath, "Media/Textures/Characters/Death/GreenFrog.png");
00331     mTextures.load(Textures::PinkFrogDeath, "Media/Textures/Characters/Death/PinkFrog.png");
00332     mTextures.load(Textures::YellowFrogDeath, "Media/Textures/Characters/Death/YellowFrog.png");
```

```
00333     mTextures.load(Textures::SpeedUp, "Media/Textures/SpeedUp.png");
00334     mTextures.load(Textures::SlowDown, "Media/Textures/SlowDown.png");
00335     mTextures.load(Textures::Freezing, "Media/Textures/Effect/Freezing/Freezing.png");
00336     mTextures.load(Textures::IceCream, "Media/Textures/IceCream.png");
00337     //mTextures.load(Textures::Honey, "Media/Textures/Honey.png");
00338     mTextures.load(Textures::Burning, "Media/Textures/Effect/Burning/Burning3.png");
00339 
00340     mTextures.load(Textures::Raining, "Media/Textures/Raining.png");
00341     mTextures.load(Textures::Snowing, "Media/Textures/Snowing.png");
00342     mTextures.load(Textures::Heart, "Media/Textures/Heart.png");
00343     mTextures.load(Textures::Animal1, "Media/Textures/" + typeMap + "/Animal/Animal1.png");
00344     mTextures.load(Textures::Animal2, "Media/Textures/" + typeMap + "/Animal/Animal2.png");
00345     mTextures.load(Textures::Animal3, "Media/Textures/" + typeMap + "/Animal/Animal3.png");
00346     mTextures.load(Textures::Animal4, "Media/Textures/" + typeMap + "/Animal/Animal4.png");
00347 }
00348 void World::loadAnimations(){
00349     std::vector<std::pair<int, sf::Vector2i» listIn4(setAnimation(typeOfMap));
00350 
00351     Animation& animal1 = mAnimation[Animations::ID::Animal1];
00352     animal1.setTexture(mTextures.get(Textures::ID::Animal1));
00353     animal1.setNumFrames(listIn4[0].first);
00354     animal1.setFrameSize(listIn4[0].second);
00355     animal1.setRepeating(true);
00356     animal1.setDuration(sf::seconds(1));
00357     animal1.scale(Constants::GridSize / animal1.getLocalBounds().width, Constants::GridSize /
      animal1.getLocalBounds().height);
00358     animal1.setOrigin(animal1.getLocalBounds().width / 2.f, animal1.getLocalBounds().height / 2.f);
00359 
00360     Animation& animal2 = mAnimation[Animations::ID::Animal2];
00361     animal2.setTexture(mTextures.get(Textures::ID::Animal2));
00362     animal2.setNumFrames(listIn4[1].first);
00363     animal2.setFrameSize(listIn4[1].second);
00364     animal2.setRepeating(true);
00365     animal2.setDuration(sf::seconds(1));
00366     animal2.scale(Constants::GridSize / animal2.getLocalBounds().width, Constants::GridSize /
      animal2.getLocalBounds().height);
00367     animal2.setOrigin(animal2.getLocalBounds().width / 2.f, animal2.getLocalBounds().height / 2.f);
00368 
00369     Animation& animal3 = mAnimation[Animations::ID::Animal3];
00370     animal3.setTexture(mTextures.get(Textures::ID::Animal3));
00371     animal3.setNumFrames(listIn4[2].first);
00372     animal3.setFrameSize(listIn4[2].second);
00373     animal3.setRepeating(true);
00374     animal3.setDuration(sf::seconds(1));
00375     animal3.scale(Constants::GridSize / animal3.getLocalBounds().width, Constants::GridSize /
      animal3.getLocalBounds().height);
00376     animal3.setOrigin(animal3.getLocalBounds().width / 2.f, animal3.getLocalBounds().height / 2.f);
00377 
00378     Animation& animal4 = mAnimation[Animations::ID::Animal4];
00379     animal4.setTexture(mTextures.get(Textures::ID::Animal4));
00380     animal4.setNumFrames(listIn4[3].first);
00381     animal4.setFrameSize(listIn4[3].second);
00382     animal4.setRepeating(true);
00383     animal4.setDuration(sf::seconds(1));
00384     animal4.scale(Constants::GridSize / animal4.getLocalBounds().width, Constants::GridSize /
      animal4.getLocalBounds().height);
00385     animal4.setOrigin(animal4.getLocalBounds().width / 2.f, animal4.getLocalBounds().height / 2.f);
00386 }
00387 void World::buildScene() {
00388     // Initialize the different layers
00389     for (std::size_t i = 0; i < LayerCount; ++i) {
00390         SceneNode::Ptr layer(new SceneNode());
00391         mSceneLayers[i] = layer.get();
00392 
00393         mSceneGraph.attachChild(std::move(layer));
00394     }
00395 
00396     // Grid making
00397     sf::Vector2f gridspawn = mSpawnPosition;
00398     gridspawn.y += Constants::initialShift * Constants::GridSize;
00399     SceneNode::Ptr grid(new GameObject(gridspawn, std::bind(&World::getBattlefieldBounds, this),
      &mTextures, mAnimation));
00400     mSceneLayers[Background]->attachChild(std::move(grid));
00401     mOriginGrid = mSpawnPosition;
00402 
00403     // Add player's character
00404 
00405     std::unique_ptr<Character> player(new Character(Character::Player, mTextures));
00406     mPlayerCharacter = player.get();
00407     mPlayerCharacter->setPosition(mSpawnPosition);
00408     mSceneLayers[Air]->attachChild(std::move(player));
00409     mPlayerCharacter->setDefaultTemperature(Constants::defaultTemperatureSpring);
00410     mPlayerCharacter->setTemperature(Constants::defaultTemperatureSpring);
00411     mPlayerCharacter->setWorldSceneGraph(&mSceneGraph);
00412 }
00413 
00414 void World::adaptPlayerPosition() {
```

```
00415     if (!mPlayerCharacter->getBoundingRect().intersects(getViewBounds())) {
00416         Command command;
00417         command.category = Category::PlayerCharacter;
00418         command.action = derivedAction<Character>([](Character& c, sf::Time) { c.destroy(); });
00419         CommandQueue& commands = getCommandQueue();
00420         commands.push(command);
00421         return;
00422     }
00423     return;
00424 }
00425
00426
00427 void World::adaptPlayerVelocity() {
00428     return;
00429 }
00430
00431 bool matchesCategories(SceneNode* node, Category::Type type) {
00432     return (node->getCategory() & type) != 0;
00433 }
00434
00435 void World::handleCollisions() {
00436     std::set<SceneNode*> playerCollidingNodes;
00437     mSceneGraph.checkNodeCollision(mPlayerCharacter->getBoundingRect(), playerCollidingNodes);
00438     for (auto node : playerCollidingNodes) {
00439         if (matchesCategories(node, Category::Obstacle)) {
00440             Command command;
00441             command.category = Category::PlayerCharacter;
00442             command.action = derivedAction<Character>([](Character& c, sf::Time) { c.destroy(); });
00443             mCommandQueue.push(command);
00444         }
00445         if (matchesCategories(node, Category::Hot)) {
00446             mPlayerCharacter->shiftTemperature(Constants::HotTemperatureShift);
00447         }
00448         if (matchesCategories(node, Category::Cold)) {
00449             mPlayerCharacter->shiftTemperature(Constants::ColdTemperatureShift);
00450         }
00451         if (matchesCategories(node, Category::SpeedUp)) {
00452             mPlayerCharacter->multSpeedMult(Constants::SpeedUpMult);
00453         }
00454         if (matchesCategories(node, Category::SlowDown)) {
00455             mPlayerCharacter->multSpeedMult(Constants::SlowDownMult);
00456         }
00457         if (matchesCategories(node, Category::HealSmall)) {
00458             mPlayerCharacter->hurt(-Constants::healAmountSmall);
00459         }
00460         if (matchesCategories(node, Category::HealLarge)) {
00461             mPlayerCharacter->hurt(-Constants::healAmountLarge);
00462         }
00463         if (matchesCategories(node, Category::HurtSmall)) {
00464             mPlayerCharacter->hurt(Constants::hurtAmountSmall);
00465         }
00466         if (matchesCategories(node, Category::HurtLarge)) {
00467             mPlayerCharacter->hurt(Constants::hurtAmountLarge);
00468         }
00469         if (matchesCategories(node, Category::PickUp)) {
00470             node->setDestroy();
00471         }
00472     }
00473 }
00474 sf::FloatRect World::getViewBounds() const
00475 {
00476     return sf::FloatRect(mWorldView.getCenter() - mWorldView.getSize() / 2.f, mWorldView.getSize());
00477 }
00478 sf::FloatRect World::getBattlefieldBounds() const
00479 {
00480     // Return view bounds + some area at top, where enemies spawn
00481     sf::FloatRect bounds = getViewBounds();
00482     bounds.top -= Constants::battlefieldBoundsHeightOffset;
00483     bounds.height += Constants::battlefieldBoundsHeightOffset*2;
00484     bounds.left -= Constants::battlefieldBoundsWidthOffset;
00485     bounds.width += Constants::battlefieldBoundsWidthOffset*2;
00486
00487     return bounds;
00488 }
00489
00490 void World::speedUp() {
00491     mPlayerCharacter->setSpeedMult(2.f);
00492 }
00493
00494 void World::slowDown() {
00495     mPlayerCharacter->setSpeedMult(0.8f);
00496 }
00497
00498 bool World::checkCState(int x) {
00499     return x&charState;
00500 }
00501
```

```
00502 void World::setWeather(int x) {
00503     weatherState=x;
00504 }
00505
00506 void World::clearWeather() {
00507     weatherState=0;
00508 }
00509
00510 bool World::isWeather(int x) {
00511     return weatherState&x;
00512 }
00513
00514 bool World::checkLastWeatherState(int x) {
00515     return lastWeatherState&x;
00516 }
```

Index