Flappy Bird

Generated by Doxygen 1.8.17

Chapter 1

Namespace Documentation

1.1 Resources Namespace Reference

Wide resource identifiers inside the game.

Enumerations

- enum Textures {
  Bird, **Bird_Spritesheet**, **Pipe_Green**, **Background**,
  Ground, **Instructions**, **Logo_Label**, **Game_Over_Label**,
  Get_Ready_Label, **Panel_Score**, **Medal_Bronze**, **Medal_Gold**,
  Medal_Platinum, **Medal_Silver** }

  The identifiers of almost all textures in the game are stored here.
- enum Fonts { **Flappy_Font** }

  The identifiers of almost all fonts in the game are stored here.

1.1.1 Detailed Description

Wide resource identifiers inside the game.

Here you will find the identifiers of the various resources, which are managed by any (x)Manager.

1.1.2 Enumeration Type Documentation

1.1.2.1 Fonts

`enum Resources::Fonts`

The identifiers of almost all fonts in the game are stored here.

As the fonts are kept in the map Resources::Fonts to std::unique_ptr<sf::Font> we use the identifiers to easily pull out the font we need.

The FontManager that allows you to do this should be available in most of the code.

1.1.2.2 Textures

```
enum Resources::Textures  [strong]
```

The identifiers of almost all textures in the game are stored here.

As the textures are kept in the map Resources::Textures to std::unique_ptr<sf::Texture> we use the identifiers to easily pull out the textures we need.

The TextureManager that allows you to do this should be available in most of the code.

Chapter 2

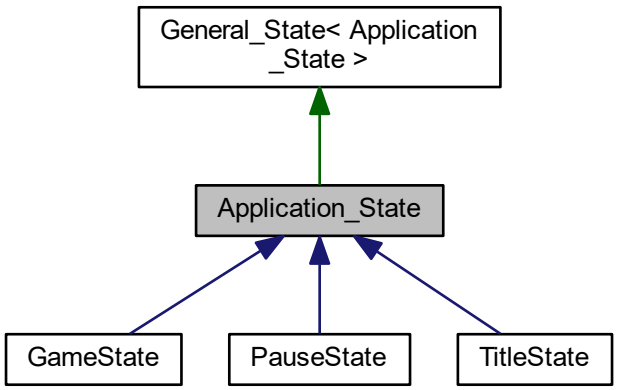
Class Documentation

2.1 Application_State Class Reference

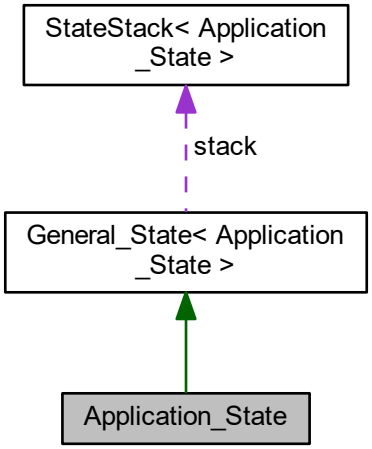
Application that controls flow of the game.

```
#include <Application_State.h>
```

Inheritance diagram for Application_State:



Collaboration diagram for Application_State:



Classes

- struct Game_Data

  Informations we want to send through states of the stack This struct is used to generate the StateStack template function.

Public Types

- typedef std::unique_ptr< Application_State > Pointer

  Type we should use in context of States.

Public Member Functions

- Application_State (StateStack< Application_State > &stack, Game_Data game_data)

  Initializes variables.
- virtual ~Application_State ()

  Declared for the purpose of correct implementation of the inheritance.
- virtual bool update (sf::Time dt)=0

  Updates the status of this state.
- virtual void draw ()=0

  Draws this state.
- virtual bool handleEvent (const sf::Event &event)=0

  Handles all events for this state.

Protected Member Functions

- Game_Data getGameData () const

Additional Inherited Members

2.1.1 Detailed Description

Application that controls flow of the game.

Main application state. Allows to switch between the states you would expect in the application itself. This are not related with game itself, but this is a state more related to the application, which only mediates with the game.

For example: it won't control behaviour of the bird itself - as it is game object. But it allows us to control behaviour of the application. Go from Title Screen to Game Screen, then to Menu, and also pause a game.

It passes further RenderWindow, TextureManager and FontManager.

2.1.2 Member Typedef Documentation

2.1.2.1 Pointer

```
typedef std::unique_ptr<Application_State> Application_State::Pointer
```

Type we should use in context of States.

This is essential for an inheritance from the General_State class.

2.1.3 Constructor & Destructor Documentation

2.1.3.1 Application_State()

```
Application_State::Application_State (
            StateStack< Application_State > & stack,
            Game_Data game_data )
```

Initializes variables.

Parameters

stack	The state stack to which this state belongs.
game_data	Data to be transmitted to the states

2.1.4 Member Function Documentation

2.1.4.1 getGameData()

```
Game_Data Application_State::getGameData ( ) const  [protected]
```

Returns

Informations that those states store for all states

2.1.4.2 handleEvent()

```
virtual bool Application_State::handleEvent (
            const sf::Event & event )  [pure virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements General_State< Application_State >.

Implemented in GameState, TitleState, and PauseState.

2.1.4.3 update()

```
virtual bool Application_State::update (
            sf::Time dt )  [pure virtual]
```

Updates the status of this state.

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements General_State< Application_State >.

Implemented in GameState, TitleState, and PauseState.

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

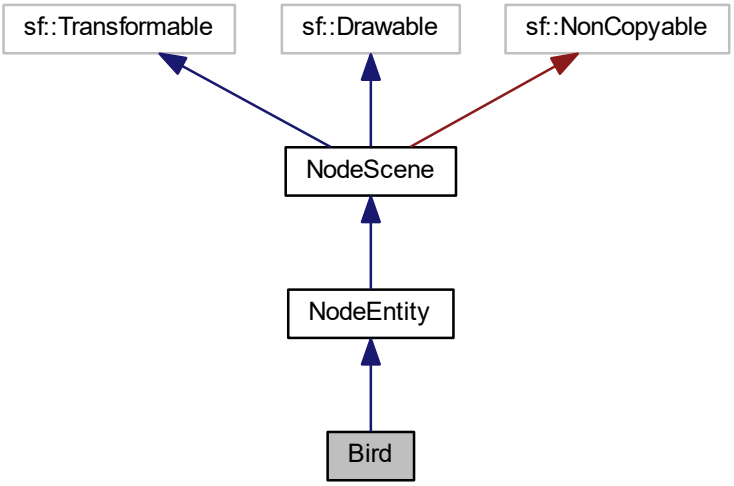
- Flappy Bird/States/Application_State.h

2.2 Bird Class Reference

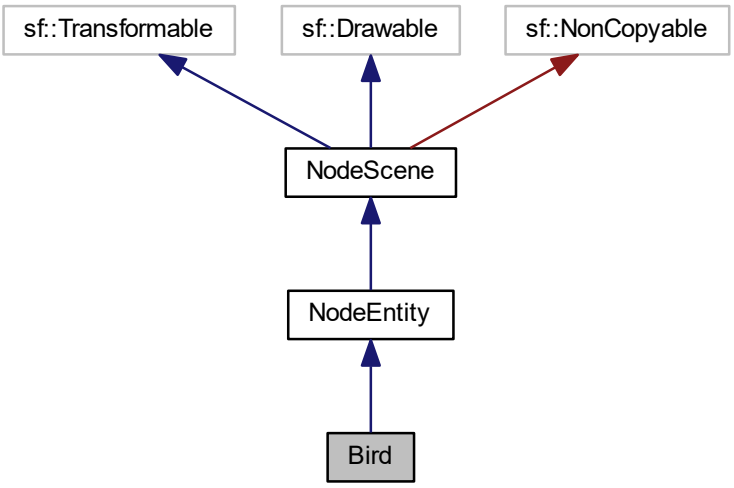
The main object of the bird, which the player controls.

`#include <Bird.h>`

Inheritance diagram for Bird:



Collaboration diagram for Bird:



Public Member Functions

- Bird (const TextureManager &textures)

  Sets the basic values of the bird, as well as reads its spritesheet and pushes individual animation frames into the deque.
- void setCollision (unsigned int col)

  It sets what kind of collision the bird has at that moment.
- void gainScore ()

  Increments the value of the result.
- virtual unsigned int getCategory () const

  Returns a category of this specific node used for signals and colisions.
- sf::FloatRect getBoundingRect () const

  Give the Rect of the object – a certain box surrounding this object in size.
- void startAnimate ()

  Starts the animation.
- void stopAnimate ()

  Stops the animation.

Friends

- class **FlappingState**
- class **FlyingState**
- class **GroundState**
- class **HitState**
- class **ScoreState**

Additional Inherited Members

2.2.1 Detailed Description

The main object of the bird, which the player controls.

There are many things about control and the player himself, as well as things about the bird itself.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Bird()

```
Bird::Bird (
            const TextureManager & textures )  [explicit]
```

Sets the basic values of the bird, as well as reads its spritesheet and pushes individual animation frames into the deque.

Parameters

textures	The Texture Manager that holds all the textures

2.2.3 Member Function Documentation

2.2.3.1 getBoundingRect()

```
sf::FloatRect Bird::getBoundingRect ( ) const  [virtual]
```

Give the Rect of the object – a certain box surrounding this object in size.

Returns

The Rect of this object

Reimplemented from NodeScene.

2.2.3.2 getCategory()

```
virtual unsigned int Bird::getCategory ( ) const  [virtual]
```

Returns a category of this specific node used for signals and colisions.

Returns

Category of this object

Reimplemented from NodeScene.

2.2.3.3 setCollision()

```
void Bird::setCollision (
            unsigned int col )
```

It sets what kind of collision the bird has at that moment.

Parameters

col	Information about collision

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

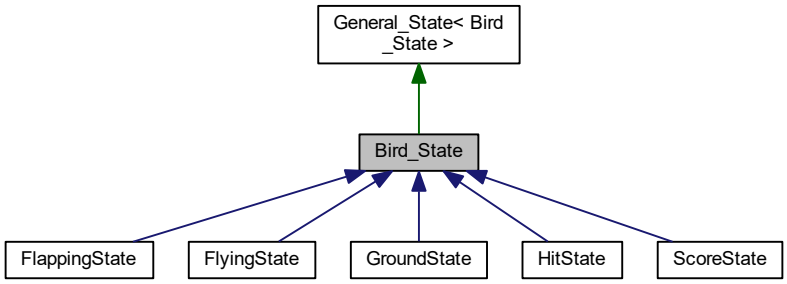
- Flappy Bird/Nodes/Specified Nodes/Bird.h

2.3 Bird_State Class Reference

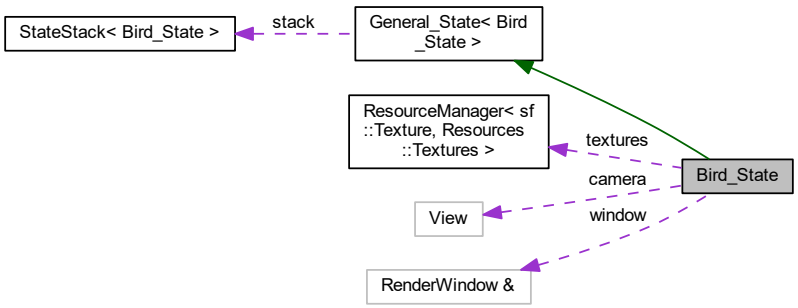
State of bird behaviour.

`#include <Bird_State.h>`

Inheritance diagram for Bird_State:



Collaboration diagram for Bird_State:



Classes

- struct Game_Data

  Informations we want to send through states of the stack This struct is used to generate the StateStack template function.

Public Types

- typedef std::unique_ptr< Bird_State > Pointer

  Type we should use in context of States.

Public Member Functions

- Bird_State (StateStack< Bird_State > &stack, Game_Data game_data)

  Initializes variables and loads the textures needed in this stack.
- virtual ~Bird_State ()

  Declared for the purpose of correct implementation of the inheritance.
- virtual bool update (sf::Time dt)=0

  Updates the status of this state.
- virtual void draw ()=0

  Draws this state.
- virtual bool handleEvent (const sf::Event &event)=0

  Handles all events for this state.
- void loadTextures ()

  Loads the textures used inside this state of the stack.

Protected Member Functions

- Game_Data getGameData () const

Protected Attributes

- TextureManager textures

  The manager that holds the textures.
- sf::RenderWindow & window

  Window to which we render the image / display the game.
- sf::View camera

  The camera we look through at the game.

2.3.1 Detailed Description

State of bird behaviour.

This state allows to control behaviour of a specific in-game object. Fully developed to break down the bird's behaviour into individual parts. In-game bird has a specific flow, for example: 1) It starts with Flying State waiting for user input 2) As it gets input it switches to Flapping State that applies gravity and allows player to "flap" the bird with another click. 3) It can now go to Hit State – when it hit a pipe, or go to Ground State – when it hits ground. Both states take away the player's ability to control the bird. The state of impact additionally forces the bird to fall down and go to Ground State. 4) Ground State displays game over, and after a while switches to Score State 5) In Score State it displays the score, and wait for player input so it can restart the game.

Warning

It is created inside GameState, so actually the main StateStack created in window class still controls the whole application with this stack itself. It means if there is Application_State pushed onto the stack it has the priority.

2.3.2 Member Typedef Documentation

2.3.2.1 Pointer

```
typedef std::unique_ptr<Bird_State> Bird_State::Pointer
```

Type we should use in context of States.

This is essential for an inheritance from the General_State class.

2.3.3 Constructor & Destructor Documentation

2.3.3.1 Bird_State()

```
Bird_State::Bird_State (
            StateStack< Bird_State > & stack,
            Game_Data game_data )
```

Initializes variables and loads the textures needed in this stack.

Parameters

stack	The state stack to which this state belongs.
game_data	Data to be transmitted to the states

2.3.4 Member Function Documentation

2.3.4.1 getGameData()

```
Game_Data Bird_State::getGameData ( ) const  [protected]
```

Returns

Informations that those states store for all states

2.3.4.2 handleEvent()

```
virtual bool Bird_State::handleEvent (
            const sf::Event & event )  [pure virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements General_State< Bird_State >.

Implemented in FlappingState, ScoreState, FlyingState, GroundState, and HitState.

2.3.4.3 update()

```
virtual bool Bird_State::update (
            sf::Time dt )  [pure virtual]
```

Updates the status of this state.

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements General_State< Bird_State >.

Implemented in FlappingState, ScoreState, FlyingState, GroundState, and HitState.

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

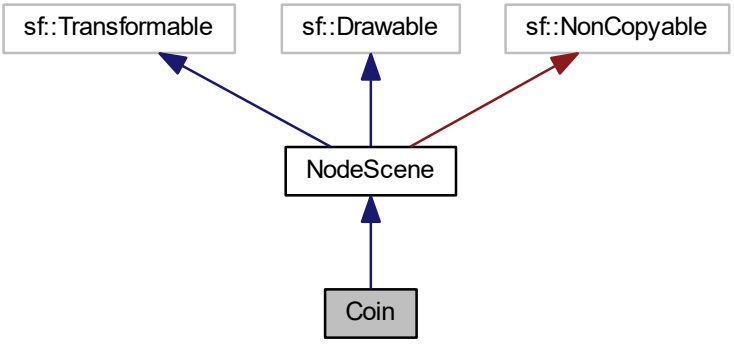
- Flappy Bird/States/Bird_State.h

2.4 Coin Class Reference

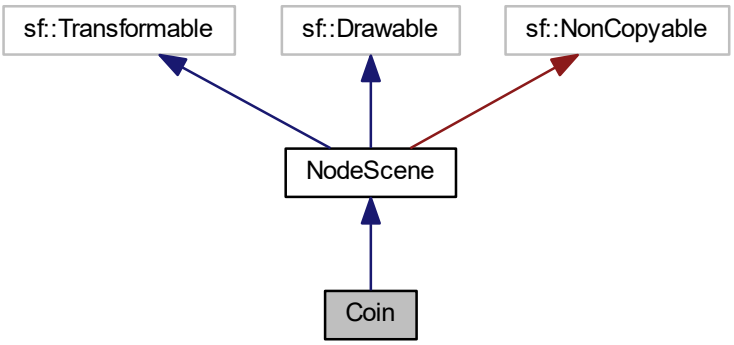
Invisible coin adding a point after collection.

```
#include <Coin.h>
```

Inheritance diagram for Coin:



Collaboration diagram for Coin:



Public Member Functions

- Coin (sf::Vector2f vec, Bird &thePlayer)

  Standard object initialization and setting up.
- sf::FloatRect getBoundingRect () const

  Give the Rect of the object – a certain box surrounding this object in size.
- void setPosition (sf::Vector2f vec)

  This function sets the positions of this object.

Additional Inherited Members

2.4.1 Detailed Description

Invisible coin adding a point after collection.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 Coin()

```
Coin::Coin (
            sf::Vector2f vec,
            Bird & thePlayer )
```

Standard object initialization and setting up.

Parameters

vec	Vector that says where this object should be placed
thePlayer	Reference to the object of the bird, which the player controls.

2.4.3 Member Function Documentation

2.4.3.1 getBoundingRect()

```
sf::FloatRect Coin::getBoundingRect ( ) const  [virtual]
```

Give the Rect of the object – a certain box surrounding this object in size.

Returns

The Rect of this object

Reimplemented from NodeScene.

2.4.3.2 setPosition()

```
void Coin::setPosition (
            sf::Vector2f vec )
```

This function sets the positions of this object.

Parameters

vec	Position at which the object is to be positioned

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

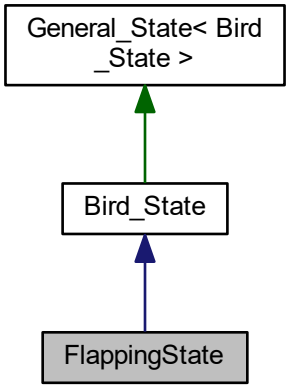
- Flappy Bird/Nodes/Specified Nodes/Coin.h

2.5 FlappingState Class Reference

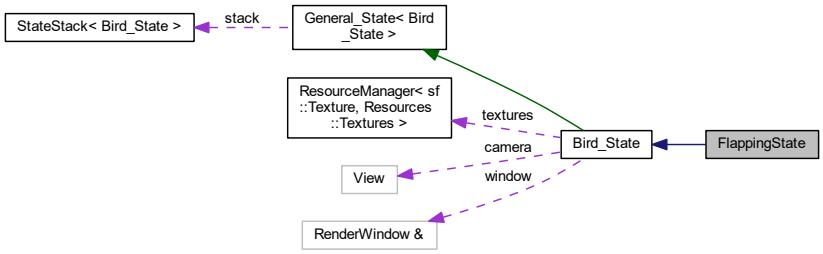
The state of play when the player controls the bird.

`#include <FlappingState.h>`

Inheritance diagram for FlappingState:



Collaboration diagram for FlappingState:



Public Member Functions

- FlappingState (StateStack< Bird_State > &statestack, Game_Data gamedata)

  It prepares the interactive part of the game, in which the player fights for the score.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.5.1 Detailed Description

The state of play when the player controls the bird.

In this state gravity acts on the bird and notoriously pushes it down. The player can interact with the bird and "flap" it.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 FlappingState()

```
FlappingState::FlappingState (
            StateStack< Bird_State > & statestack,
            Game_Data gamedata )
```

It prepares the interactive part of the game, in which the player fights for the score.

It sends a signal to the pipe generator to start working. It also sets the position where the result is displayed. It also initializes the variables.

Parameters

statestack	The state stack to which this state belongs.
gamedata	Data to be transmitted to the states

2.5.3 Member Function Documentation

2.5.3.1 handleEvent()

```
virtual bool FlappingState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Bird_State.

2.5.3.2 update()

```
virtual bool FlappingState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Bird_State.

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

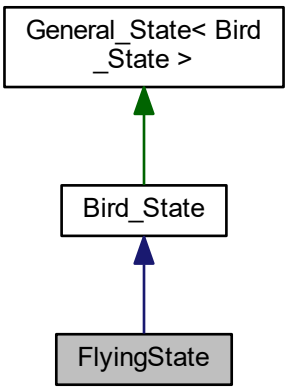
- Flappy Bird/States/Bird States/FlappingState.h

2.6 FlyingState Class Reference

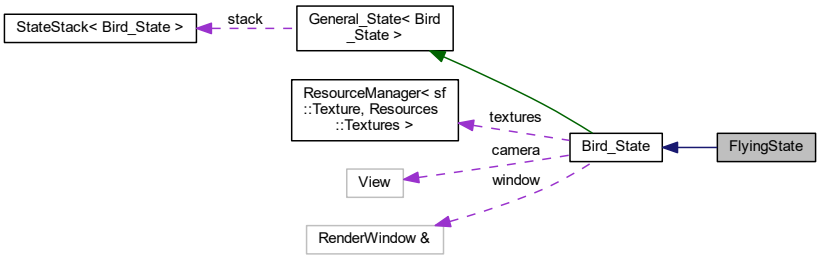
A state where bird is flying and waiting for user interaction.

`#include <FlyingState.h>`

Inheritance diagram for FlyingState:



Collaboration diagram for FlyingState:



Public Member Functions

- FlyingState (StateStack< Bird_State > &statestack, Game_Data gamedata)

  Loads necessary textures and initializes variables.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.6.1 Detailed Description

A state where bird is flying and waiting for user interaction.

The state in which the bird is waiting in the air for the player's first flap.

In this state, incoming pipes are not yet generated. The bird flies up and down and the screen shows instructions on how to make the first jump, and waits for interaction with the player.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 FlyingState()

```
FlyingState::FlyingState (
            StateStack< Bird_State > & statestack,
            Game_Data gamedata )
```

Loads necessary textures and initializes variables.

Parameters

statestack	The state stack to which this state belongs.
gamedata	Data to be transmitted to the states

2.6.3 Member Function Documentation

2.6.3.1 handleEvent()

```
virtual bool FlyingState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Bird_State.

2.6.3.2 update()

```
virtual bool FlyingState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Bird_State.

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

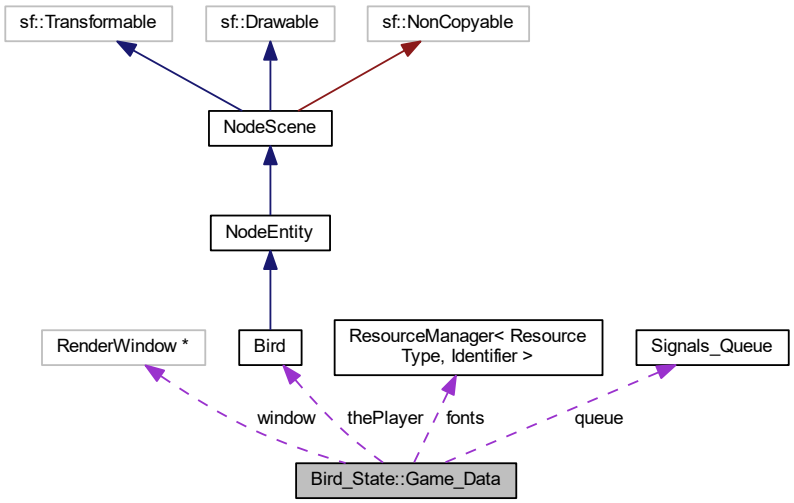
- Flappy Bird/States/Bird States/FlyingState.h

2.7 Bird_State::Game_Data Struct Reference

Informations we want to send through states of the stack This struct is used to generate the StateStack template function.

`#include <Bird_State.h>`

Collaboration diagram for Bird_State::Game_Data:



Public Member Functions

- Game_Data (sf::RenderWindow &window, Signals_Queue &queue, FontManager &fonts, Bird *&bird)

  Initializes all data we want to transfer to the states in the stack.

Public Attributes

- sf::RenderWindow * window

  Window we draw image into.
- FontManager * fonts

  Font holder that holds fonts to use.
- Signals_Queue * queue

  Pointer to queue of signals.
- Bird *& thePlayer

  Reference to pointer to the object of the bird, which the player controls.

2.7.1 Detailed Description

Informations we want to send through states of the stack This struct is used to generate the StateStack template function.

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

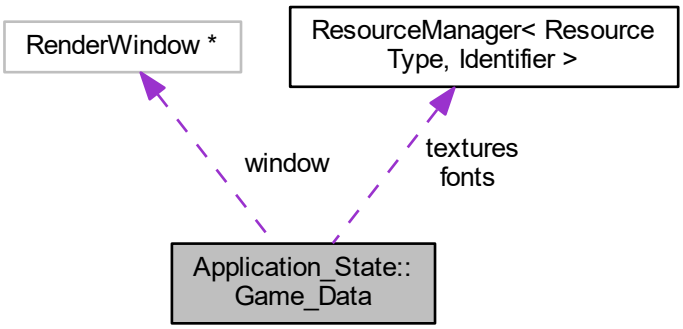
- Flappy Bird/States/Bird_State.h

2.8 Application_State::Game_Data Struct Reference

Informations we want to send through states of the stack This struct is used to generate the StateStack template function.

`#include <Application_State.h>`

Collaboration diagram for Application_State::Game_Data:



Public Member Functions

- Game_Data (sf::RenderWindow &window, TextureManager &textures, FontManager &fonts)

  Initializes all data we want to transfer to the states in the stack.

Public Attributes

- sf::RenderWindow * window

  Window we draw image into.
- TextureManager * textures

  Texture Holder that holds textures to use.
- FontManager * fonts

  Font holder that holds fonts to use.

2.8.1 Detailed Description

Informations we want to send through states of the stack This struct is used to generate the StateStack template function.

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

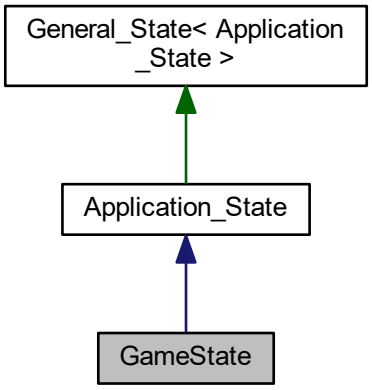
- Flappy Bird/States/Application_State.h

2.9 GameState Class Reference

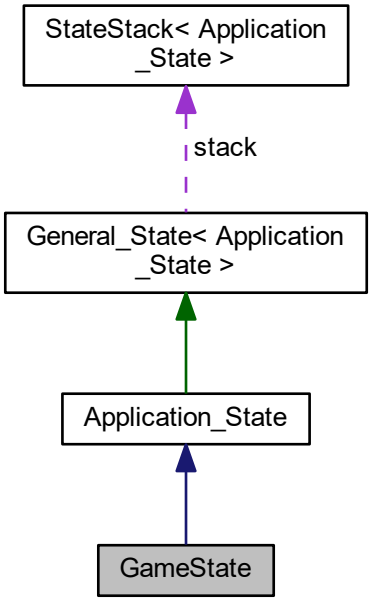
A state where the actual game starts.

`#include <GameState.h>`

Inheritance diagram for GameState:



Collaboration diagram for GameState:



Public Member Functions

- GameState (StateStack< Application_State > &stack, Game_Data game_data)

  Initializes variables and prepares the game state.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.
- void loadFonts (FontManager &fonts)

  Loads the fonts used inside this state.

Additional Inherited Members

2.9.1 Detailed Description

A state where the actual game starts.

The target game that the player can interact with and have fun.

The whole mechanics of the game takes place here. The game world and its objects are kept in this state. There is also another StateStack here which controls flow of states of the bird.

2.9.2 Constructor & Destructor Documentation

2.9.2.1 GameState()

```
GameState::GameState (
            StateStack< Application_State > & stack,
            Game_Data game_data )
```

Initializes variables and prepares the game state.

It loads fonts, also loads states to the StateStack bird, which is controlled by the player. It sets the default state of the StateStack of the bird to "Flying".

Parameters

stack	The state stack to which this state belongs.
game_data	Data to be transmitted to the states

2.9.3 Member Function Documentation

2.9.3.1 handleEvent()

```
virtual bool GameState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Application_State.

2.9.3.2 loadFonts()

```
void GameState::loadFonts (
            FontManager & fonts )
```

Loads the fonts used inside this state.

Parameters

fonts	The font manager to which we load the fonts

2.9.3.3 update()

```
virtual bool GameState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Application_State.

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

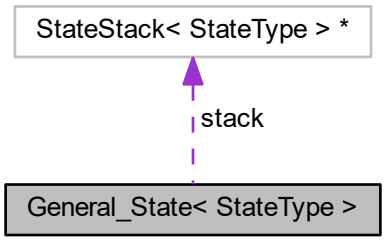
- Flappy Bird/States/Game States/GameState.h

2.10 General_State< StateType > Class Template Reference

An abstract General_State other states can inherit from.

`#include <General_State.h>`

Collaboration diagram for General_State< StateType >:



Public Member Functions

- General_State (StateStack< StateType > &stack)
  Only the statestack is being initialized.
- virtual ~General_State ()
  Declared for the purpose of correct implementation of the inheritance.
- virtual bool update (sf::Time dt)=0
  Updates the status of this state.
- virtual void draw ()=0
  Draws this state.
- virtual bool handleEvent (const sf::Event &event)=0
  Handles all events for this state.

Protected Member Functions

- void requestStackPush (States::ID state)
  Sends a request to the stack to push the state to the stack.
- void requestStackPop ()
  Sends a request to the stack to pop state (from the top) from the stack.
- void requestStackClear ()
  Sends a request to clear the stack.

Protected Attributes

- StateStack< StateType > * stack

  Pointer to the stack to which this state belongs.

2.10.1 Detailed Description

template<typename StateType>
class General_State< StateType >

An abstract General_State other states can inherit from.

This abstract class defines all general behaviours of the State like for example: 1) request Push – push given state onto the stack 2) request Pop – pops state from the top of the stack 3) request Clear – clears the stack

Also it defines some typical SFML functions like: update, draw or handle events.

Template Parameters

StateType	type of the State so it can be a General State of specfic StateStack of given type.

2.10.2 Constructor & Destructor Documentation

2.10.2.1 General_State()

```
template<typename StateType >
General_State< StateType >::General_State (
            StateStack< StateType > & stack )
```

Only the statestack is being initialized.

Parameters

stack	The stack that stores this state

2.10.3 Member Function Documentation

2.10.3.1 handleEvent()

```
template<typename StateType >
virtual bool General_State< StateType >::handleEvent (
            const sf::Event & event )  [pure virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implemented in Bird_State, Application_State, GameState, TitleState, PauseState, FlappingState, ScoreState, FlyingState, GroundState, and HitState.

2.10.3.2 requestStackClear()

```
template<typename StateType >
void General_State< StateType >::requestStackClear  [protected]
```

Sends a request to clear the stack.

Ask for the clear – but it can't do it instantly, because it would be program breaking, so it request for it to be done in next frame. The request lands in the queue, which will be read in the next loop run.

2.10.3.3 requestStackPop()

```
template<typename StateType >
void General_State< StateType >::requestStackPop  [protected]
```

Sends a request to the stack to pop state (from the top) from the stack.

Ask for the pop – but it can't do it instantly, because it would be program breaking, so it request for it to be done in next frame. The request lands in the queue, which will be read in the next loop run.

2.10.3.4 requestStackPush()

```
template<typename StateType >
void General_State< StateType >::requestStackPush (
            States::ID state )  [protected]
```

Sends a request to the stack to push the state to the stack.

Ask for the push – but it can't do it instantly, because it would be program breaking, so it request for it to be done in next frame. The request lands in the queue, which will be read in the next loop run.

Parameters

state	The identifier of the state to push onto the stack.

2.10.3.5 update()

```
template<typename StateType >
virtual bool General_State< StateType >::update (
            sf::Time dt )  [pure virtual]
```

Updates the status of this state.

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implemented in Bird_State, Application_State, GameState, TitleState, PauseState, FlappingState, ScoreState, FlyingState, GroundState, and HitState.

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

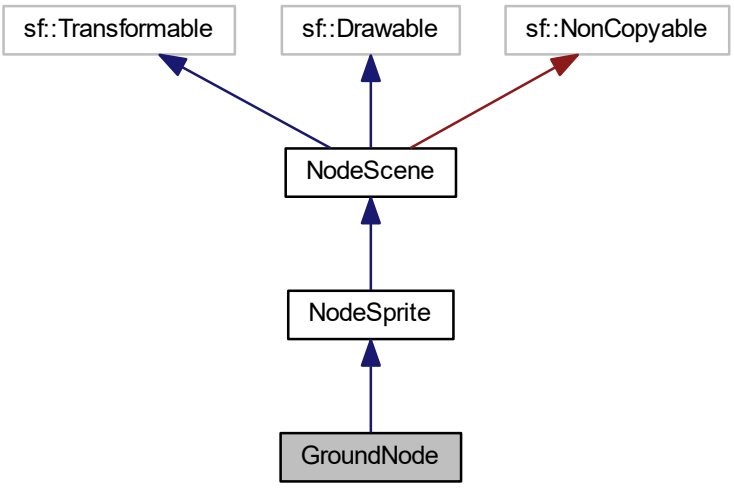
- Flappy Bird/States/General_State.h

2.11 GroundNode Class Reference

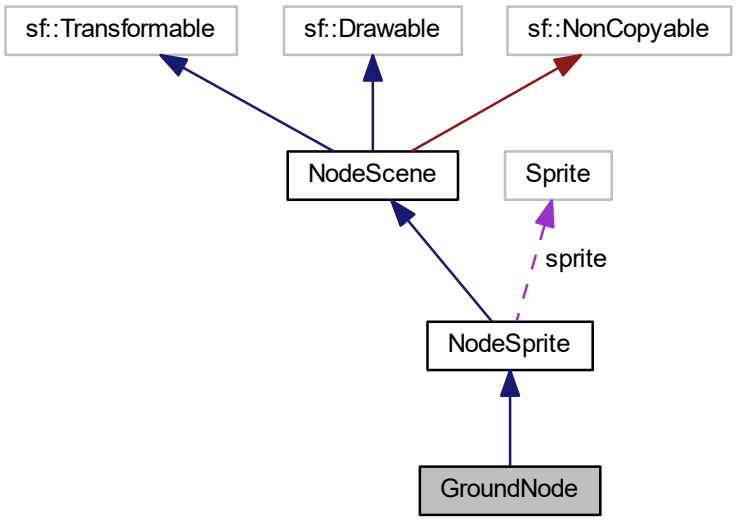
A node of the floor, which moves continuously to the left.

`#include <GroundNode.h>`

Inheritance diagram for GroundNode:



Collaboration diagram for GroundNode:



Public Member Functions

- GroundNode (TextureManager &textures, const sf::IntRect &rect)

  It loads the object's texture, and sets its infinite repetition.

Additional Inherited Members

2.11.1 Detailed Description

A node of the floor, which moves continuously to the left.

When it is far enough away from the screen, it will return to its natural position, thus pretending the endless movement of the bird to the right.

2.11.2 Constructor & Destructor Documentation

2.11.2.1 GroundNode()

```
GroundNode::GroundNode (
            TextureManager & textures,
            const sf::IntRect & rect )
```

It loads the object's texture, and sets its infinite repetition.

Parameters

textures	The Texture Manager that holds textures
rect	The rect size of this object (size of the sprite)

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

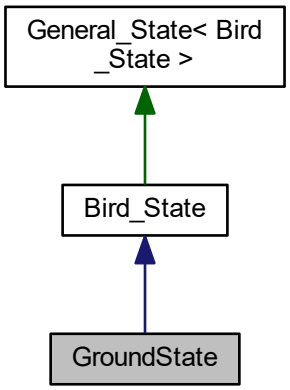
- Flappy Bird/Nodes/Specified Nodes/GroundNode.h

2.12 GroundState Class Reference

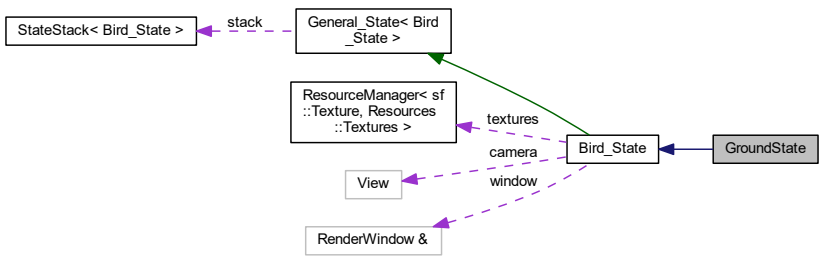
This is the condition in which the bird hit the ground.

`#include <GroundState.h>`

Inheritance diagram for GroundState:



Collaboration diagram for GroundState:



Public Member Functions

- GroundState (StateStack< Bird_State > &statestack, Game_Data gamedata)

  It loads the necessary texture and sets the "game over" sign.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.12.1 Detailed Description

This is the condition in which the bird hit the ground.

In this state the player cannot control the bird. This state displays "Game Over", and waits a few seconds after which it changes State to "ScoreState".

2.12.2 Constructor & Destructor Documentation

2.12.2.1 GroundState()

```
GroundState::GroundState (
            StateStack< Bird_State > & statestack,
            Game_Data gamedata )
```

It loads the necessary texture and sets the "game over" sign.

Parameters

statestack	The state stack to which this state belongs.
gamedata	Data to be transmitted to the states

2.12.3 Member Function Documentation

2.12.3.1 handleEvent()

```
virtual bool GroundState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Bird_State.

2.12.3.2 update()

```
virtual bool GroundState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Bird_State.

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

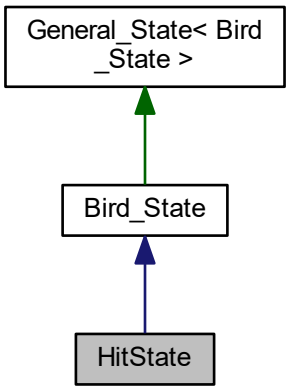
- Flappy Bird/States/Bird States/GroundState.h

2.13 HitState Class Reference

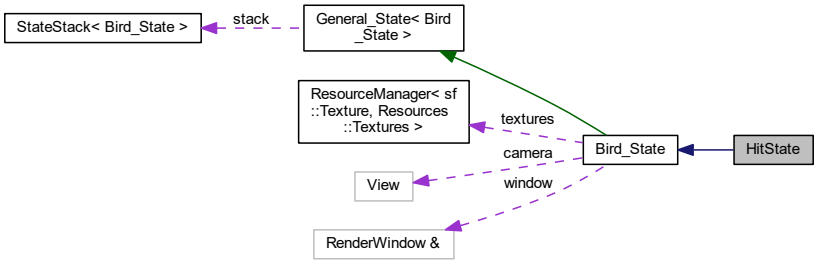
This is a condition that starts when a player hits a pipe with a bird.

```
#include <HitState.h>
```

Inheritance diagram for HitState:



Collaboration diagram for HitState:



Public Member Functions

- HitState (StateStack< Bird_State > &statestack, Game_Data gamedata)

  Initializes variables.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.13.1 Detailed Description

This is a condition that starts when a player hits a pipe with a bird.

The player is deprived of all bird control here. The bird is facing the ground and moving at high speed towards it.

2.13.2 Constructor & Destructor Documentation

2.13.2.1 HitState()

```
HitState::HitState (
            StateStack< Bird_State > & statestack,
            Game_Data gamedata )
```

Initializes variables.

Parameters

statestack	The state stack to which this state belongs.
gamedata	Data to be transmitted to the states

2.13.3 Member Function Documentation

2.13.3.1 handleEvent()

```
virtual bool HitState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Bird_State.

2.13.3.2 update()

```
virtual bool HitState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Bird_State.

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

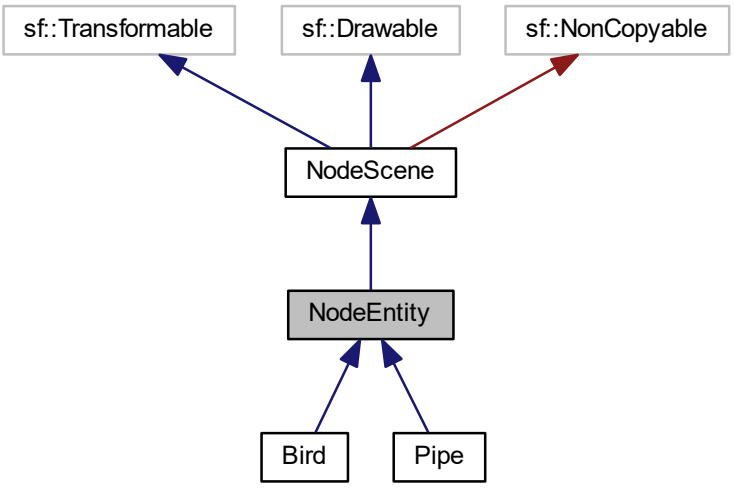
- Flappy Bird/States/Bird States/HitState.h

2.14 NodeEntity Class Reference

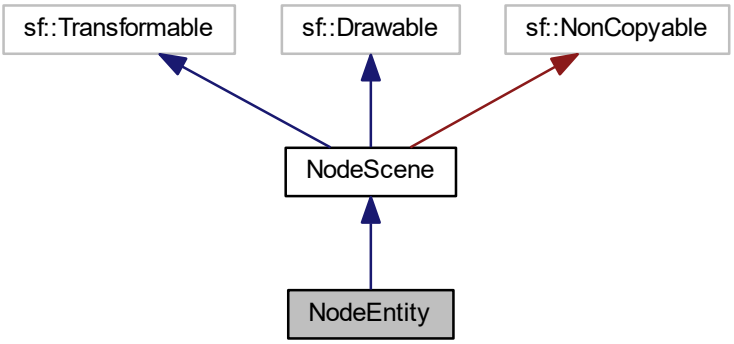
The kind of scene that contains movement A specific type of scene that contains physical elements such as acceleration, speed.

```
#include <NodeEntity.h>
```

Inheritance diagram for NodeEntity:



Collaboration diagram for NodeEntity:



Public Member Functions

- void setVelocity (sf::Vector2f vel)

  Set velocity of this object.
- void setVelocity (float vx, float vy)

  Set velocity of this object.
- sf::Vector2f getVelocity () const
- NodeEntity & operator+= (sf::Vector2f velocity)

  Adds velocity to current velocity.
- NodeEntity & operator-= (sf::Vector2f velocity)

  Decrease velocity to current velocity.
- void accelerate (sf::Vector2f velocity)

  Adds velocity to current velocity.

Additional Inherited Members

2.14.1 Detailed Description

The kind of scene that contains movement A specific type of scene that contains physical elements such as acceleration, speed.

2.14.2 Member Function Documentation

2.14.2.1 accelerate()

```
void NodeEntity::accelerate (
            sf::Vector2f velocity )
```

Adds velocity to current velocity.

Parameters

velocity	Velocity we want to add to this object

2.14.2.2 getVelocity()

```
sf::Vector2f NodeEntity::getVelocity ( ) const
```

Returns

Velocity of this object

2.14.2.3 operator+=()

```
NodeEntity& NodeEntity::operator+= (
            sf::Vector2f velocity )
```

Adds velocity to current velocity.

Returns

itself

2.14.2.4 operator-=()

```
NodeEntity& NodeEntity::operator-= (
            sf::Vector2f velocity )
```

Decrease velocity to current velocity.

Returns

itself

2.14.2.5 setVelocity() [1/2]

```
void NodeEntity::setVelocity (
            float vx,
            float vy )
```

Set velocity of this object.

Parameters

vx	Velocity on the x-axis
vy	Velocity on the y-axis

2.14.2.6 setVelocity() [2/2]

```
void NodeEntity::setVelocity (
            sf::Vector2f vel )
```

Set velocity of this object.

Parameters

vel	Vector with velocity applied to this object

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

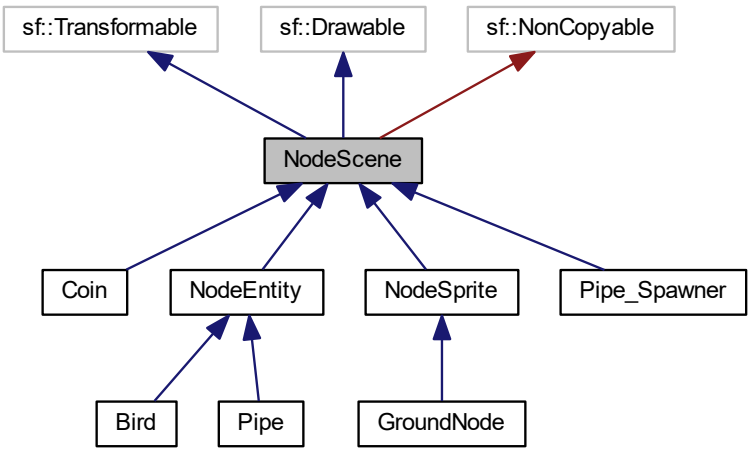
- Flappy Bird/Nodes/NodeEntity.h

2.15 NodeScene Class Reference

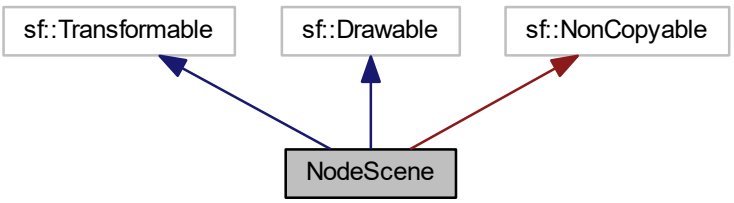
The class is a kind of "canvas" on which we can draw certain objects and combine them.

`#include <NodeScene.h>`

Inheritance diagram for NodeScene:



Collaboration diagram for NodeScene:



Public Types

- typedef std::unique_ptr< NodeScene > Node

  Easier naming on Pointer for this type.

Public Member Functions

- NodeScene ()

  Initializes the scene with default values.
- NodeScene (const NodeScene &)=delete

  I don't want copy-assignment in that type.
- NodeScene (NodeScene &&node) noexcept

  Move constructor.
- NodeScene & operator= (NodeScene &&node) noexcept

  Move assignment operator.
- void pin_Node (Node node)

  It "pins" to itself another node, in other words it attaches it as its child.
- Node unpin_Node (const NodeScene &node)

  It "unpins" from itself another node, in other words it deattaches its child.
- void removePinnedNodes ()

  Remove all pinned nodes to this node.
- sf::Transform GetAbsoluteTransform () const

  It gets absolute transform relative to the window – not relative to the scene to which it is attached.
- void update (sf::Time dt)

  Updates the status of this node scene (informations in it)
- void interpretSignal (const Signal &signal, sf::Time dt)

  Checks if the signal is for his category, and if so then it interprets it.
- virtual sf::FloatRect getBoundingRect () const

  Give the Rect of the object – a certain box surrounding this object in size.
- unsigned int checkNodeCollision (Bird &bird)

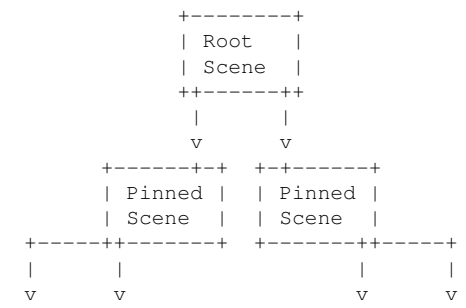
  Check collision only with the bird (thats how flappy bird works)

2.15.1 Detailed Description

The class is a kind of "canvas" on which we can draw certain objects and combine them.

NodeScene works like a bit more complex canvas and is called a scene for a reason. First of all, you can draw and manage certain objects on the scenes. You can clip other scenes to the scene. Scenes resemble a kind of tree, which can perform operations on scenes that are pinned to them. Also, scenes pinned to another subordinate scene move in relation to the subordinate scene.

Scenes can transmit a signal downwards, so that the signal will be interpreted by each of them. It is enough that only one main scene is updated, or drawn, and all the scenes pinned to it will also be.



+---—+-+ +-+---—+ +---—+-+ +-+---—+ | Pinned | | Pinned | | Pinned | | Pinned | | Scene | | Scene | | Scene | | Scene | +-------—+ +-------—+ +-------—+ +-------—+

2.15.2 Constructor & Destructor Documentation

2.15.2.1 NodeScene()

```
NodeScene::NodeScene (
            NodeScene && node )  [noexcept]
```

Move constructor.

Parameters

node	Node we should steal values from

2.15.3 Member Function Documentation

2.15.3.1 checkNodeCollision()

```
unsigned int NodeScene::checkNodeCollision (
            Bird & bird )
```

Check collision only with the bird (thats how flappy bird works)

Parameters

bird	Reference the object of the bird, which the player controls.

Returns

Information on types of collisions

2.15.3.2 GetAbsoluteTransform()

```
sf::Transform NodeScene::GetAbsoluteTransform ( ) const
```

It gets absolute transform relative to the window – not relative to the scene to which it is attached.

Returns

The global transform of this object

2.15.3.3 getBoundingRect()

```
virtual sf::FloatRect NodeScene::getBoundingRect ( ) const  [virtual]
```

Give the Rect of the object – a certain box surrounding this object in size.

Returns

The Rect of this object

Reimplemented in Bird, Pipe, NodeSprite, and Coin.

2.15.3.4 interpretSignal()

```
void NodeScene::interpretSignal (
            const Signal & signal,
            sf::Time dt )
```

Checks if the signal is for his category, and if so then it interprets it.

Parameters

signal	Signal to be interpreted
dt	The time elapsed between the previous and the new frame.

2.15.3.5 operator=()

```
NodeScene& NodeScene::operator= (
            NodeScene && node )  [noexcept]
```

Move assignment operator.

Parameters

node	Node we should steal values from

2.15.3.6 pin_Node()

```
void NodeScene::pin_Node (
            Node node )
```

It "pins" to itself another node, in other words it attaches it as its child.

Parameters

node	The node we want to pin to this node.

2.15.3.7 unpin_Node()

```
Node NodeScene::unpin_Node (
            const NodeScene & node )
```

It "unpins" from itself another node, in other words it deattaches its child.

Parameters

node	The node we want to unpin from this node.

Returns

The unpinned node

2.15.3.8 update()

```
void NodeScene::update (
            sf::Time dt )
```

Updates the status of this node scene (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

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

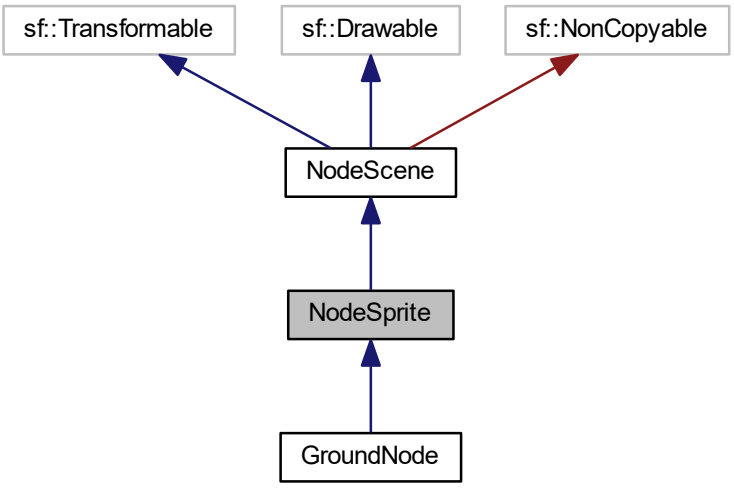
- Flappy Bird/Nodes/NodeScene.h

2.16 NodeSprite Class Reference

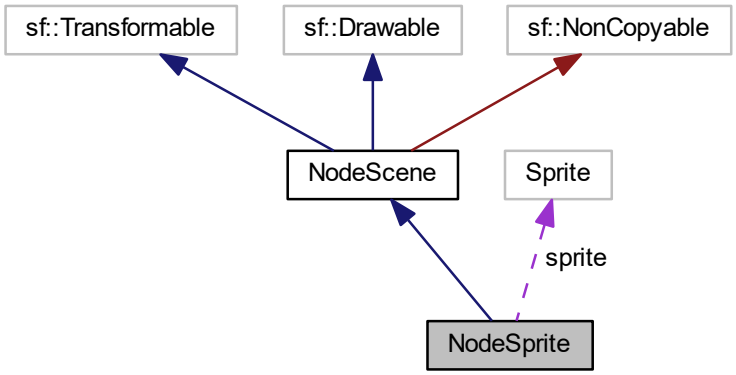
The kind of scene that includes "Sprite" and the operations associated with it.

```
#include <NodeSprite.h>
```

Inheritance diagram for NodeSprite:



Collaboration diagram for NodeSprite:



Public Member Functions

- NodeSprite (const sf::Texture &texture)

  Initializes the texture of this object.
- NodeSprite (const sf::Texture &texture, const sf::IntRect &rect)

  Initializes the texture and rect of this object.
- virtual sf::FloatRect getLocalBounds ()
- virtual sf::FloatRect getBoundingRect () const

  Give the Rect of the object – a certain box surrounding this object in size.

Protected Attributes

- sf::Sprite sprite

  Image of this object.

Additional Inherited Members

2.16.1 Detailed Description

The kind of scene that includes "Sprite" and the operations associated with it.

This kind of scene has some kind of graphical robe that we want to draw or manage.

2.16.2 Constructor & Destructor Documentation

2.16.2.1 NodeSprite() [1/2]

```
NodeSprite::NodeSprite (
            const sf::Texture & texture )  [explicit]
```

Initializes the texture of this object.

Parameters

texture	The Texture Manager that holds the textures

2.16.2.2 NodeSprite() [2/2]

```
NodeSprite::NodeSprite (
            const sf::Texture & texture,
            const sf::IntRect & rect )
```

Initializes the texture and rect of this object.

Parameters

texture	The Texture Manager that holds the textures
rect	The rect size of this object (size of the sprite)

2.16.3 Member Function Documentation

2.16.3.1 getBoundingRect()

```
virtual sf::FloatRect NodeSprite::getBoundingRect ( ) const  [virtual]
```

Give the Rect of the object – a certain box surrounding this object in size.

Returns

The Rect of this object

Reimplemented from NodeScene.

2.16.3.2 getLocalBounds()

```
virtual sf::FloatRect NodeSprite::getLocalBounds ( )  [virtual]
```

Returns

local boundaries around this object

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

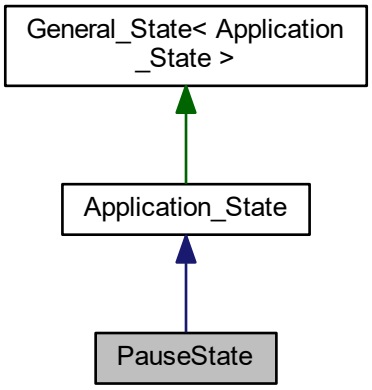
- Flappy Bird/Nodes/NodeSprite.h

2.17 PauseState Class Reference

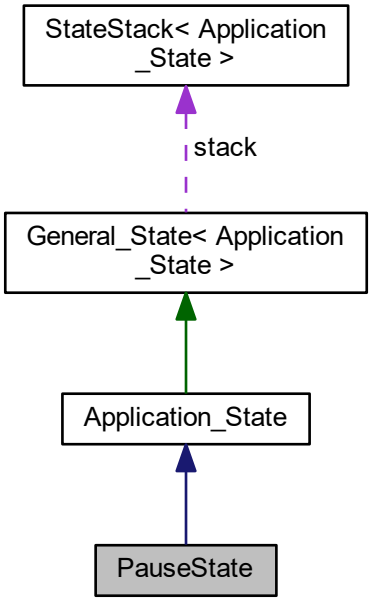
A state that allows the player to stop the game and then resume it again or leave the game.

`#include <PauseState.h>`

Inheritance diagram for PauseState:



Collaboration diagram for PauseState:



Public Member Functions

- PauseState (StateStack< Application_State > &state, Game_Data game_data)

  Initializes variables and prepares a graphical pause screen scene.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.17.1 Detailed Description

A state that allows the player to stop the game and then resume it again or leave the game.

This state is rather superimposed on other states. It draws a semi-transparent and dark colour on the screen, and displays information about the game state and how to get out.

It also blocks the states that have been superimposed underneath it, so that time does not flow in the game.

2.17.2 Constructor & Destructor Documentation

2.17.2.1 PauseState()

```
PauseState::PauseState (
            StateStack< Application_State > & state,
            Game_Data game_data )
```

Initializes variables and prepares a graphical pause screen scene.

Parameters

state	The state stack to which this state belongs.
game_data	Data to be transmitted to the states

2.17.3 Member Function Documentation

2.17.3.1 handleEvent()

```
virtual bool PauseState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Application_State.

2.17.3.2 update()

```
virtual bool PauseState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Application_State.

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

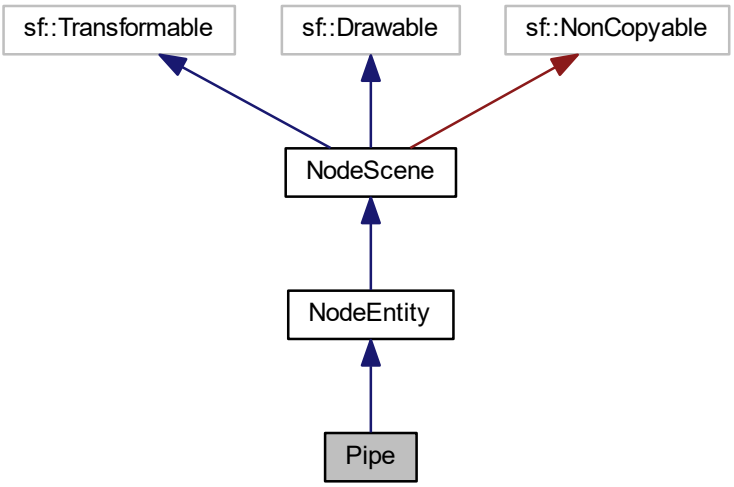
- Flappy Bird/States/Game States/PauseState.h

2.18 Pipe Class Reference

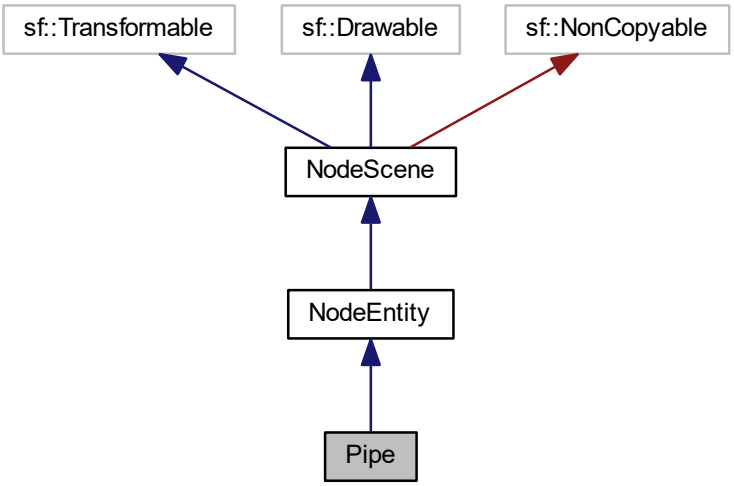
A pipe is an object that can move.

```
#include <Pipe.h>
```

Inheritance diagram for Pipe:



Collaboration diagram for Pipe:



Public Types

- enum Type { **Green**, **Red** }

  Different types of pipes were planned in the game, but have not yet been implemented.

Public Member Functions

- Pipe (const TextureManager &textures, Type type=Type::Green)

  Standardly initializes the pipe.
- virtual unsigned int getCategory () const

  Returns a category of this specific node used for signals and colisions.
- virtual void drawThisNode (sf::RenderTarget &target, sf::RenderStates states) const

  Function that prints only elements related to this node.
- sf::FloatRect getBoundingRect () const

  Give the Rect of the object – a certain box surrounding this object in size.

Friends

- class **TypicalPipe**

2.18.1 Detailed Description

A pipe is an object that can move.

It has a category for which a collision is displayed.

2.18.2 Constructor & Destructor Documentation

2.18.2.1 Pipe()

```
Pipe::Pipe (
            const TextureManager & textures,
            Type type = Type::Green )
```

Standardly initializes the pipe.

Parameters

textures	The Texture Manager that holds the textures
type	Type of the pipe

2.18.3 Member Function Documentation

2.18.3.1 drawThisNode()

```
virtual void Pipe::drawThisNode (
            sf::RenderTarget & target,
```

```
sf::RenderStates states ) const  [virtual]
```

Function that prints only elements related to this node.

Parameters

target	Canvas on which we draw all objects
states	Any transformations and additional effects (in our case, transformations relative to the scene)

Reimplemented from NodeScene.

2.18.3.2 getBoundingRect()

```
sf::FloatRect Pipe::getBoundingRect ( ) const  [virtual]
```

Give the Rect of the object – a certain box surrounding this object in size.

Returns

The Rect of this object

Reimplemented from NodeScene.

2.18.3.3 getCategory()

```
virtual unsigned int Pipe::getCategory ( ) const  [virtual]
```

Returns a category of this specific node used for signals and colisions.

Returns

Category of this object

Reimplemented from NodeScene.

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

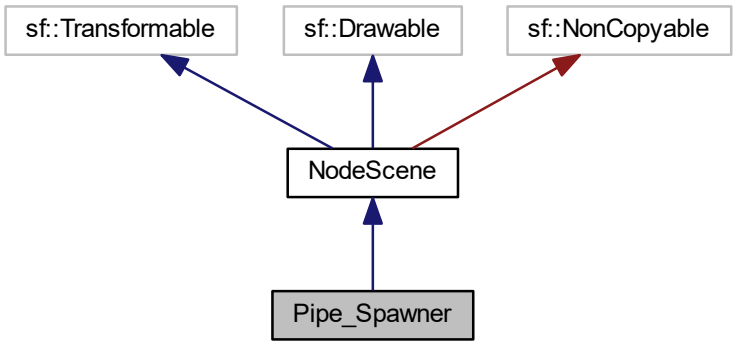
- Flappy Bird/Nodes/Specified Nodes/Pipe.h

2.19 Pipe_Spawner Class Reference

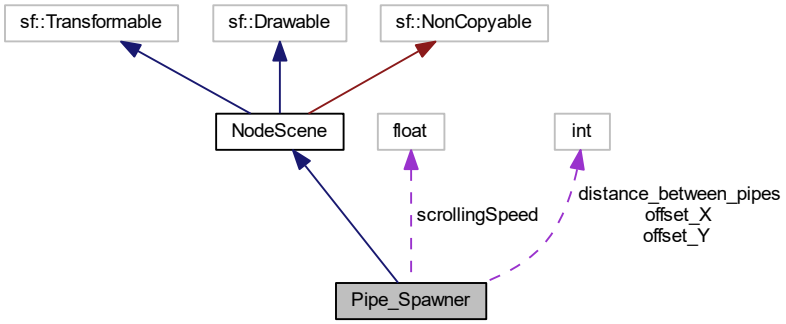
A pipe generator which, based on the given parameters, generates pipes at appropriate speed and at appropriate intervals.

`#include <Pipe_Spawner.h>`

Inheritance diagram for Pipe_Spawner:



Collaboration diagram for Pipe_Spawner:



Public Member Functions

- Pipe_Spawner (const TextureManager &textures, sf::Vector2f spawn_position, Bird &thePlayer)

  Initializes all variables.
- void start_spawning ()

  This function makes the generator start to generate pipes.
- void stop_spawning ()

  This function makes the generator stop to generate pipes.
- virtual unsigned int getCategory () const override

  Returns a category of this specific node used for signals and colisions.

Protected Attributes

- int offset_Y = 60

  What should be the space between the pipes in the Y axis.
- int offset_X = 25

  What space should be on X axis between Pipes.
- int distance_between_pipes = 35

  Minimal distance between pipes.
- float scrollingSpeed = 30.2f

  How fast spawned pipes move.

Additional Inherited Members

2.19.1 Detailed Description

A pipe generator which, based on the given parameters, generates pipes at appropriate speed and at appropriate intervals.

Pipe Spawner should have been placed in (0,0) (left up corner of the screen)

2.19.2 Constructor & Destructor Documentation

2.19.2.1 Pipe_Spawner()

```
Pipe_Spawner::Pipe_Spawner (
            const TextureManager & textures,
            sf::Vector2f spawn_position,
            Bird & thePlayer )
```

Initializes all variables.

Parameters

textures	The Texture Manager that hold the textures
spawn_position	Position at which the pipes should be generated
thePlayer	Reference to the object of the bird, which the player controls.

2.19.3 Member Function Documentation

2.19.3.1 getCategory()

```
virtual unsigned int Pipe_Spawner::getCategory ( ) const  [override], [virtual]
```

Returns a category of this specific node used for signals and colisions.

Returns

Category of this object

Reimplemented from NodeScene.

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

- Flappy Bird/Nodes/Specified Nodes/Pipe_Spawner.h

2.20 ResourceManager< ResourceType, Identifier > Class Template Reference

This class allows us to hold some textures in it, and get them back later.

`#include <ResourceManager.h>`

Public Member Functions

- void load_resource (Identifier group, const std::string &file)

  Loads the resource from the file, and stores it under given identifier name.
- template<typename Param >
  void load_resource (Identifier group, const std::string &file, const Param &param)

  Loads the resource from the file, and stores it under given identifier name.
- ResourceType & get_resource (Identifier id)

  Gives us reference to the resource of given identifier.
- const ResourceType & get_resource (Identifier id) const

  Gives us reference to the resource of given identifier.

2.20.1 Detailed Description

template<typename ResourceType, typename Identifier>
class ResourceManager< ResourceType, Identifier >

This class allows us to hold some textures in it, and get them back later.

I use this function due to the fact that Textures, or sounds are very heavy, and I want to store them once, and them use them multiple times.

Template Parameters

ResourceType	Type of the resource we want to store
Identifier	The identifiers we want to use when selecting back to our resources.

2.20.2 Member Function Documentation

2.20.2.1 get_resource() [1/2]

```
template<typename ResourceType , typename Identifier >
ResourceType& ResourceManager< ResourceType, Identifier >::get_resource (
            Identifier id )
```

Gives us reference to the resource of given identifier.

Parameters

id	Resource identifier to be read

Returns

Reference to the resource of given identifier

2.20.2.2 get_resource() [2/2]

```
template<typename ResourceType , typename Identifier >
const ResourceType& ResourceManager< ResourceType, Identifier >::get_resource (
            Identifier id ) const
```

Gives us reference to the resource of given identifier.

Parameters

id	Resource identifier to be read

Returns

Reference to the resource of given identifier

2.20.2.3 load_resource() [1/2]

```
template<typename ResourceType , typename Identifier >
void ResourceManager< ResourceType, Identifier >::load_resource (
            Identifier group,
            const std::string & file )
```

Loads the resource from the file, and stores it under given identifier name.

Parameters

group	The identifier of the resource under whose name you want to save and read the object.
file	File from which we read a given resource

2.20.2.4 load_resource() [2/2]

```
template<typename ResourceType , typename Identifier >
template<typename Param >
void ResourceManager< ResourceType, Identifier >::load_resource (
            Identifier group,
            const std::string & file,
            const Param & param )
```

Loads the resource from the file, and stores it under given identifier name.

Template Parameters

Param	Additional type that you may want to use when storing the resource.

Parameters

group	The identifier of the resource under whose name you want to save and read the object.
file	File from which we read a given resource
param	Additional parameter that you may want to use when storing the resource.

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

- Flappy Bird/Resources/ResourceManager.h

2.21 Score Class Reference

Text that displays the player's current score.

```
#include <Score.h>
```

Public Member Functions

- Score (FontManager &fonts, Resources::Fonts the_font=Resources::Fonts::Flappy_Font)

  Sets the basic values that the text displaying the score should have.
- void set_score (int score)

  Sets the score that this object displays.
- void setPosition (sf::Vector2f pos)

  Changes the position of this object displaying the score.

- void setPosition (float pos_x, float pos_y)

  Changes the position of this object displaying the score.
- void draw (sf::RenderWindow &window) const

  Function that draws this object.
- void update ()

  Updates the status of this object (informations in it)

2.21.1 Detailed Description

Text that displays the player's current score.

It is displayed in the middle of the screen and continuously updates the pipes that have been passed.

2.21.2 Constructor & Destructor Documentation

2.21.2.1 Score()

```
Score::Score (
            FontManager & fonts,
            Resources::Fonts the_font = Resources::Fonts::Flappy_Font )
```

Sets the basic values that the text displaying the score should have.

Parameters

fonts	Font Manager that holds the fonts
the_font	The font that we want to use to display the score

2.21.3 Member Function Documentation

2.21.3.1 draw()

```
void Score::draw (
            sf::RenderWindow & window ) const
```

Function that draws this object.

Parameters

window	Window on which we draw an object

2.21.3.2 set_score()

```
void Score::set_score (
            int score )
```

Sets the score that this object displays.

Parameters

score	Score to display

2.21.3.3 setPosition() [1/2]

```
void Score::setPosition (
            float pos_x,
            float pos_y )
```

Changes the position of this object displaying the score.

Parameters

pos_x	Position on the x-axis on which you want to place the object
pos_y	Position on the y-axis on which you want to place the object

2.21.3.4 setPosition() [2/2]

```
void Score::setPosition (
            sf::Vector2f pos )
```

Changes the position of this object displaying the score.

Parameters

pos	Vector containing the position on which we want to place an object

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

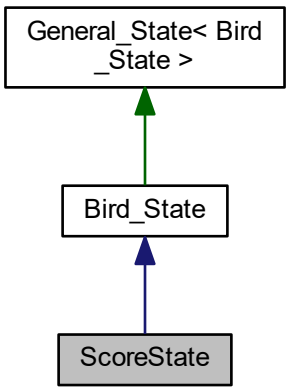
- Flappy Bird/Nodes/Specified Nodes/Score.h

2.22 ScoreState Class Reference

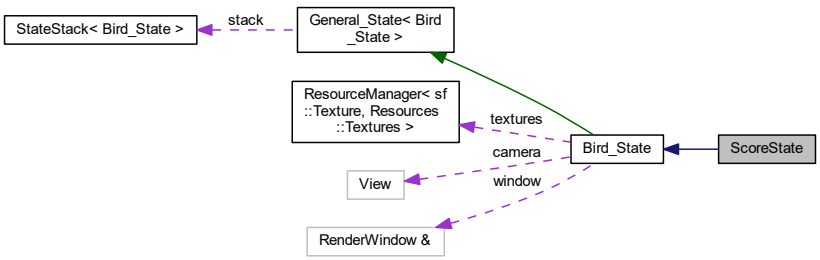
In this state, the score that the player has achieved is displayed.

`#include <ScoreState.h>`

Inheritance diagram for ScoreState:



Collaboration diagram for ScoreState:



Public Member Functions

- ScoreState (StateStack< Bird_State > &statestack, Game_Data gamedata)

  Initializes variables and prepares the player's score.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.22.1 Detailed Description

In this state, the score that the player has achieved is displayed.

It is also given the opportunity to play the game again by clicking any button.

2.22.2 Constructor & Destructor Documentation

2.22.2.1 ScoreState()

```
ScoreState::ScoreState (
            StateStack< Bird_State > & statestack,
            Game_Data gamedata )
```

Initializes variables and prepares the player's score.

It chooses after the player's score which medal should be displayed to him. It composes the whole scene with the result that the player has got.

Parameters

statestack	The state stack to which this state belongs.
gamedata	Data to be transmitted to the states

2.22.3 Member Function Documentation

2.22.3.1 handleEvent()

```
virtual bool ScoreState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Bird_State.

2.22.3.2 update()

```
virtual bool ScoreState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Bird_State.

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

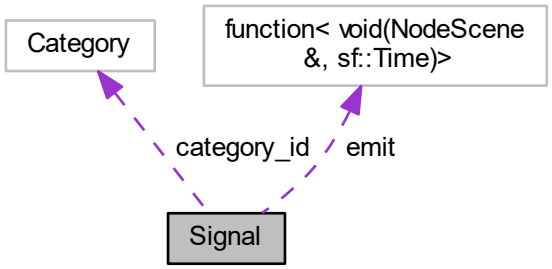
- Flappy Bird/States/Bird States/ScoreState.h

2.23 Signal Struct Reference

Signals are a special function, which is implicitly performed on elements of a given scene.

```
#include <Signal.h>
```

Collaboration diagram for Signal:



Public Member Functions

- Signal ()
  Initializes the variables.

Public Attributes

- std::function< void(NodeScene &, sf::Time)> emit
  It specifies what kind of work should the targeted node do.
- Signals::Category category_id
  Category of the signal.

2.23.1 Detailed Description

Signals are a special function, which is implicitly performed on elements of a given scene.

This works in such a way that it starts with 'root_scene', which interprets whether or not a given signal is intended to continue by using prepared for this category. If it is intended for it, it performs on itself the function contained in the signal and regardless of whether the signal was intended for it or not, it passes it on for all scenes pinned to it. They do the same.

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

- Flappy Bird/Signals/Signal.h

2.24 Signals_Queue Class Reference

This is a class that acts as a stack or queue of the "FIFO" type and allows the storage of signals.

`#include <Signals_Queue.h>`

Public Member Functions

- void push (const Signal &signal)
  Pushes the signal to the queue.
- Signal pop ()
  Pops the signal from the queue, and returns it.
- bool isEmpty () const
  Check if the queue is empty.

2.24.1 Detailed Description

This is a class that acts as a stack or queue of the "FIFO" type and allows the storage of signals.

This queue is being passed on to a huge number of game areas. It allows communication of unconnected game objects.

2.24.2 Member Function Documentation

2.24.2.1 pop()

```
Signal Signals_Queue::pop ( )
```

Pops the signal from the queue, and returns it.

Returns

The signal that was popped from the queue.

2.24.2.2 push()

```
void Signals_Queue::push (
            const Signal & signal )
```

Pushes the signal to the queue.

Parameters

signal	Signal to be pushed to the queue

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

- Flappy Bird/Signals/Signals_Queue.h

2.25 StateStack< StateType > Class Template Reference

A simpler version of finite state machine called State Stack.

```
#include <StateStack.h>
```

Public Types

- enum Operation { **Push**, **Pop**, **Clear** }
  All operations that can be performed on our stack.
- using State_Data = typename StateType::Game_Data
  Easier name for the data the stack transfers to its states.
- using State_Pointer = typename StateType::Pointer
  Easier name on the pointer for the type of state that is stacked.

Public Member Functions

- StateStack (State_Data gamedata)

  At the moment, it only initialize the variables.
- StateStack< StateType > & operator+= (States::ID state)

  Alternative possibility to perform Push operation on the stack.
- StateStack< StateType > & operator-- ()

  Alternative possibility to perform Pop operation on the stack.
- template<typename T >
  void loadState (States::ID state)

  Loads the given state, so it can easily use it later.
- void pushState (States::ID state)

  Push the state on the stack (on top of the stack)
- void popState ()

  Pop the state from the stack (from top of the stack)
- void clearStack ()

  Removes all states from the stack.
- bool isEmpty () const

  Checks if there is any state on the stack.
- std::string getLogs ()

  Returns collected logs, and cleans them.
- void update (sf::Time dt)

  Updates the status of all states inside the state stack.
- void draw ()

  Draw all states inside the stack (calls their draw() method)
- void handleEvent (const sf::Event &event)

  Handles all events inside the stack.

2.25.1 Detailed Description

template<typename StateType>
class StateStack< StateType >

A simpler version of finite state machine called State Stack.

Made according to practices in "SFML Game Development Book"

StateStack that stores states of the game.

In its overall operation, it allows it to act similarly to the behaviour of a stack. Certain states of the game can be overlapped or removed from the stack.

For example: 1) We can have TitleState which displays logo of the company. 2) Then TitleState is removed from the stack, and GameState is Pushed 3) While the game works, and player is playing we can push PauseState on the stack. Now PauseState code execute first, and it can (but do not have to) block layers of stack underneath. This way we can easily make PauseState draw transparent darkish rectangle on the screen, while GameState underneath is still working, or not if we want to block it inside PauseState.

Template Parameters

StateType	type of State we want to put on stack later on. They might define different Data, or parameters that will be passed to the stack.

2.25.2 Member Enumeration Documentation

2.25.2.1 Operation

```
template<typename StateType >
enum StateStack::Operation
```

All operations that can be performed on our stack.

Specifically, these are their identifiers, which easily allow you to send information to the stack about what operation should be performed.

2.25.3 Constructor & Destructor Documentation

2.25.3.1 StateStack()

```
template<typename StateType >
StateStack< StateType >::StateStack (
            State_Data gamedata )  [explicit]
```

At the moment, it only initialize the variables.

Parameters

gamedata	Information that is transferred to the states at the stake.

2.25.4 Member Function Documentation

2.25.4.1 getLogs()

```
template<typename StateType >
std::string StateStack< StateType >::getLogs  [inline]
```

Returns collected logs, and cleans them.

Returns

Collected logs since the last getLogs() function was called

2.25.4.2 handleEvent()

```
template<typename StateType >
void StateStack< StateType >::handleEvent (
             const sf::Event & event )
```

Handles all events inside the stack.

Calls a function that handles events in any state present in the stack.

Parameters

event	Events stored in the window.

2.25.4.3 isEmpty()

```
template<typename StateType >
bool StateStack< StateType >::isEmpty
```

Checks if there is any state on the stack.

Returns

True if there is no state on the stack

2.25.4.4 loadState()

```
template<typename StateType >
template<typename T >
void StateStack< StateType >::loadState (
             States::ID state )
```

Loads the given state, so it can easily use it later.

Under a given identifier in the States map::ID to std::function<State_Pointer()> a callable object is created, which creates a state object for us. This object is then used by createState(...) to create the state and push it onto the stack.

Template Parameters

T	A specific state class that inherits from the class that StateStack stores.

Parameters

state	The name of the identifier that you want to assign to the class of the loaded state.

2.25.4.5 operator+=()

```
template<typename StateType >
StateStack< StateType > & StateStack< StateType >::operator+= (
            States::ID state )  [inline]
```

Alternative possibility to perform Push operation on the stack.

Parameters

state	The identifier of the state to push onto the stack.

2.25.4.6 pushState()

```
template<typename StateType >
void StateStack< StateType >::pushState (
            States::ID state )
```

Push the state on the stack (on top of the stack)

Parameters

state	The identifier of the state to push onto the stack.

2.25.4.7 update()

```
template<typename StateType >
void StateStack< StateType >::update (
            sf::Time dt )
```

Updates the status of all states inside the state stack.

Parameters

dt	The time elapsed between the previous and the new frame.

The documentation for this class was generated from the following files:

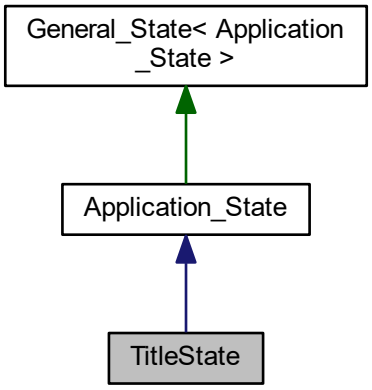
- Flappy Bird/States/General_State.h
- Flappy Bird/States/StateStack.h

2.26 TitleState Class Reference

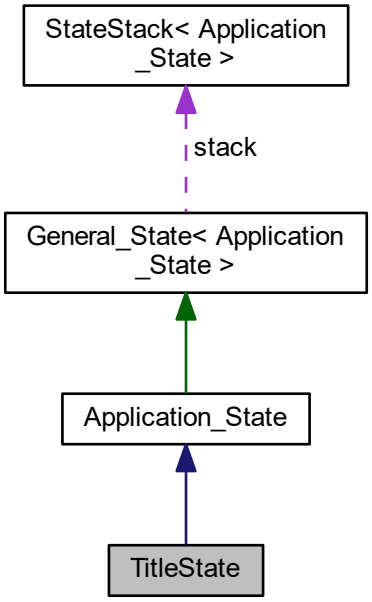
A state displaying animated image of the creator.

`#include <TitleState.h>`

Inheritance diagram for TitleState:



Collaboration diagram for TitleState:



Public Member Functions

- TitleState (StateStack< Application_State > &stack, Game_Data game_data)

  Initializes variables and extracts individual frames from the spritesheet and pushes them into the "background" vector.
- virtual void draw ()

  Draws this state.
- virtual bool update (sf::Time dt)

  Updates the status of this state (informations in it)
- virtual bool handleEvent (const sf::Event &event)

  Handles all events for this state.

Additional Inherited Members

2.26.1 Detailed Description

A state displaying animated image of the creator.

This state is pushed only at beginning of the program, as a kind of representation of the author. It displays an animated image for few seconds, and then it moves to the Game State.

2.26.2 Constructor & Destructor Documentation

2.26.2.1 TitleState()

```
TitleState::TitleState (
            StateStack< Application_State > & stack,
            Game_Data game_data )
```

Initializes variables and extracts individual frames from the spritesheet and pushes them into the "background" vector.

Parameters

stack	The state stack to which this state belongs.
game_data	Data to be transmitted to the states

2.26.3 Member Function Documentation

2.26.3.1 handleEvent()

```
virtual bool TitleState::handleEvent (
            const sf::Event & event )  [virtual]
```

Handles all events for this state.

Parameters

event	Events stored in the window.

Returns

If false, this is information to stop to handle events on the lower layers of the stack.

Implements Application_State.

2.26.3.2 update()

```
virtual bool TitleState::update (
            sf::Time dt )  [virtual]
```

Updates the status of this state (informations in it)

Parameters

dt	The time elapsed between the previous and the new frame.

Returns

If false, this is information to stop the update on the lower layers of the stack.

Implements Application_State.

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

- Flappy Bird/States/Game States/TitleState.h

2.27 Window Class Reference

This file is everything that is related to the window application - for example handle exit buttons, resizing the window etc.

```
#include <Window.h>
```

Public Member Functions

- Window ()

  It prepares the window for proper operation.
- ~Window ()

  Properly closes the log file.
- void run ()

  It launches the main processes of the game, which last for the entire duration of the game.

2.27.1 Detailed Description

This file is everything that is related to the window application - for example handle exit buttons, resizing the window etc.

A window where the game should be displayed.

The window deals with displaying the game. It does not manage anything related to the game gameplay itself, and its maximum interference with the game itself is displaying the StateStack, and loading states used in the StateStack.

It also holds Manager for Texture and Fonts – so they can be used anywhere inside the window.

2.27.2 Constructor & Destructor Documentation

2.27.2.1 Window()

```
Window::Window ( )
```

It prepares the window for proper operation.

In its current form, it creates a window with a resolution of 432 x 768 and sets the window positions to the top left of the screen. It also loads the states to the state and sets the window icon. It applies a lock of 60 frames per second. Initializes most variables.

2.27.3 Member Function Documentation

2.27.3.1 run()

```
void Window::run ( )
```

It launches the main processes of the game, which last for the entire duration of the game.

It calls functions that process inputs and update the game state. Calculates the correct time between frames and generates game status updates no faster than 60 times per second. It also controls the state of the game and closes the window when the game is finished.

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

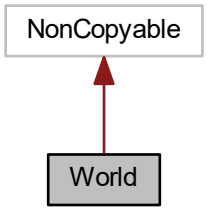
- Flappy Bird/Window.h

2.28 World Class Reference

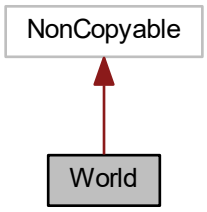
A game world class that includes everything that happens inside the game world.

`#include <World.h>`

Inheritance diagram for World:



Collaboration diagram for World:



Public Member Functions

- World (sf::RenderWindow &window, Bird *&bird)

  The designer of the game world.
- Signals_Queue & getSignalQueue ()
- void resetWorld ()

  Removes all objects in the main scene "root_scene" and creates them again.
- void update (sf::Time dt)

  Updates the status of all objects inside the game world.
- void draw ()

  Draw all objects inside the game world.
- bool handleCollisions ()

  It checks if there have been any collisions in the game world between the player and the environment.

2.28.1 Detailed Description

A game world class that includes everything that happens inside the game world.

It stores the main scene called "root_scene". Draws all objects and sets them according to the layers on the main game scene.

2.28.2 Constructor & Destructor Documentation

2.28.2.1 World()

```
World::World (
            sf::RenderWindow & window,
            Bird *& bird )
```

The designer of the game world.

In its current state, it calls up a function that reads textures, and creates a scene.

Parameters

window	The window to which we draw objects.
bird	The player's object – the bird that the player controls.

2.28.3 Member Function Documentation

2.28.3.1 getSignalQueue()

```
Signals_Queue& World::getSignalQueue ( )
```

Returns

Reference to Queue of Signals

2.28.3.2 handleCollisions()

```
bool World::handleCollisions ( )
```

It checks if there have been any collisions in the game world between the player and the environment.

It checks for collisions on each scene between players. Assigns the type of collision that occurred and assigns it to a variable inside the Bird class.

Returns

True if there are collisions between the player and the environment.

2.28.3.3 update()

```
void World::update (
            sf::Time dt )
```

Updates the status of all objects inside the game world.

Parameters

dt	The time elapsed between the previous and the new frame.

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

- Flappy Bird/World.h

Index