Happy Birds

Generated by Doxygen 1.9.8

Chapter 1

Source content

A brief summary of all classes. UML diagram showing the class relations can be found in the project documentation.

1.0.1 Game

Encapsulates the main control logic for a game and executes the game loop.

1.0.2 GUI

Manages the game's graphical user interface.

1.0.3 Level

Constructs levels according to the information of a CSV file. Includes physics simulation.

1.0.4 ReadFile

Reads the CSV file containing level information.

1.0.5 Player

Represents a user who has logged in to the game and saves scores for the user.

1.0.6 LevelEditor

Enables users to create their own levels by placing desired items to desired locations.

1.0.7 CollisionDetection

Handles collisions and contact events in the game.

1.0.8 Bird

Represents the bird in the game that is launched towards the fortress.

1.0.9 SpecialBird

Represents the bird with the special attack feature. Inherits from Bird class.

1.0.10 Pig

Represents the pig in the game, which is the enemy to be destroyed.

1.0.11 Box

Represents a movable box object in the game.

1.0.12 Wall

Represents an immovable wall object in the game.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Topic Documentation

6.1 Level Properties

Functions

- Player::Player (std::string name)

  Constructs a Player object.
- std::string Player::loadFromFile ()

  Loads player data from a file.
- void **Player::saveToFile** ()

  Saves player data to a file.
- int Player::getScoreForLevel (int levelNumber)

  Gets the score earned by the player for a specific level.
- void Player::updateScore (int levelNumber, int stars)

  Updates the score for a specific level if it's higher than the previous score.

Variables

- std::map< int, int > **Player::levelScores**

6.1.1 Detailed Description

6.1.2 Function Documentation

6.1.2.1 getScoreForLevel()

```
int Player::getScoreForLevel (
             int levelNumber )
```

Gets the score earned by the player for a specific level.

Retrieves the score for a specific level.

Parameters

levelNumber	The level number.

Returns

The score earned by the player for the specified level.

Parameters

levelNumber	The number of the level.

Returns

The score earned by the player for that level.

6.1.2.2 loadFromFile()

```
std::string Player::loadFromFile ( )
```

Loads player data from a file.

Returns

Message indicating the player status.

A message indicating whether the player is new or existing.

6.1.2.3 Player()

```
Player::Player (
             std::string name )
```

Constructs a Player object.

Parameters

name	The name of the player.

6.1.2.4 updateScore()

```
void Player::updateScore (
             int levelNumber,
             int stars )
```

Updates the score for a specific level if it's higher than the previous score.

Parameters

levelNumber	The number of the level.
stars	The number of stars earned.

6.2 SpecialBird

Defines the SpecialBird class, a type of Bird with specific shooting behavior.

Classes

- class SpecialBird

  Represents a special type of bird with specific shooting behavior.

6.2.1 Detailed Description

Defines the SpecialBird class, a type of Bird with specific shooting behavior.

Chapter 7

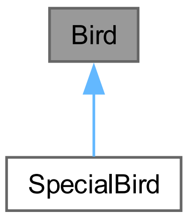
Class Documentation

7.1 Bird Class Reference

Represents a bird object in the game.

```
#include <bird.hpp>
```

Inheritance diagram for Bird:



Public Member Functions

- Bird (b2World *world, const sf::Texture &texture, const b2Vec2 &position)

  Constructor for the Bird class.
- void update ()

  Updates the bird's state.
- void render (sf::RenderWindow &window)

  Renders the bird on the provided SFML window.
- virtual void handleInput (const sf::Event &event, const sf::RenderWindow &window)

  Handles input events for the bird.
- void launch (const b2Vec2 &force)

  Launches the bird with a specified force.
- b2Vec2 getVelocity () const

Get the current velocity of the bird.
- bool isBirdLaunched () const

  Check if the bird has been launched.
- b2Body ∗ getBody () const

  Get the Box2D body of the bird.
- std::vector< sf::CircleShape > calculateTrajectory (const sf::Vector2f &launchVector, int numDots)

  Calculate the trajectory points.

Public Attributes

- b2Body ∗ **body**

  The bird's Box2D body.
- std::vector< sf::CircleShape > **trajectoryDots**

  Stores the trajectory dots.

7.1.1 Detailed Description

Represents a bird object in the game.

The Bird class encapsulates the properties and behaviors of a bird in the game environment. It integrates both the rendering and physics aspects of the bird.

This class manages the bird's shape, Box2D body, launch mechanism, input handling for dragging, rendering on an SFML window, updating its position and rotation, as well as calculating and storing trajectory dots used for aiming before launch.

The bird can be launched with a specified force and its behavior is governed by Box2D physics, including velocity, collision, and movement in the game world.

It maintains states such as whether the bird has been launched or is currently being dragged. The number of birds per level and the index of the current bird are also managed within this class.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Bird()

```
Bird::Bird (
            b2World * world,
            const sf::Texture & texture,
            const b2Vec2 & position )
```

Constructor for the Bird class.

Constructs a Bird object.

Parameters

world	Pointer to the Box2D world.
texture	The texture for the bird.
position	The initial position of the bird.

7.1.3 Member Function Documentation

7.1.3.1 calculateTrajectory()

```
std::vector< sf::CircleShape > Bird::calculateTrajectory (
             const sf::Vector2f & launchVector,
             int numDots )
```

Calculate the trajectory points.

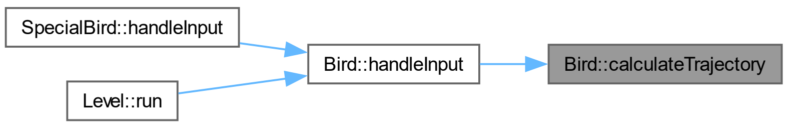
Parameters

launchVector	The vector representing the launch direction.
numDots	The number of trajectory dots to calculate.

Returns

A vector containing the calculated trajectory dots.

Here is the caller graph for this function:



7.1.3.2 getBody()

```
b2Body * Bird::getBody ( ) const
```

Get the Box2D body of the bird.

Returns

A pointer to the Box2D body of the bird.

7.1.3.3 getVelocity()

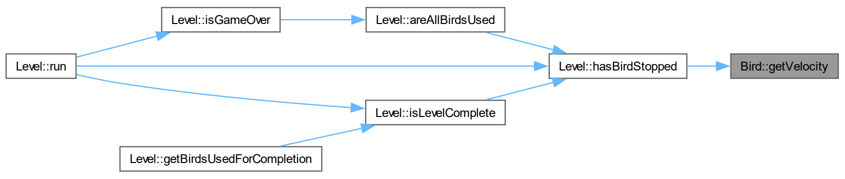
```
b2Vec2 Bird::getVelocity ( ) const
```

Get the current velocity of the bird.

Returns

The velocity vector of the bird.

Here is the caller graph for this function:



7.1.3.4 handleInput()

```
void Bird::handleInput (
             const sf::Event & event,
             const sf::RenderWindow & window )  [virtual]
```

Handles input events for the bird.

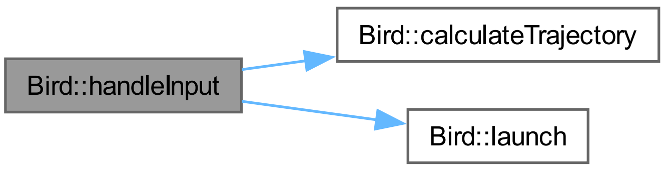
Handles input events for the bird, including dragging and launching.

Parameters

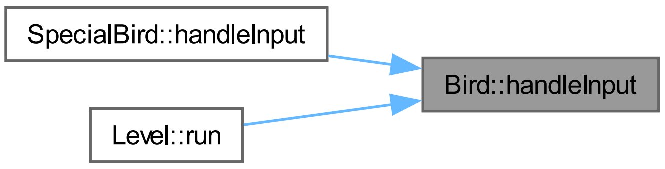
event	The SFML event to handle.
window	The SFML window associated with the event.

Reimplemented in SpecialBird.

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.5 isBirdLaunched()

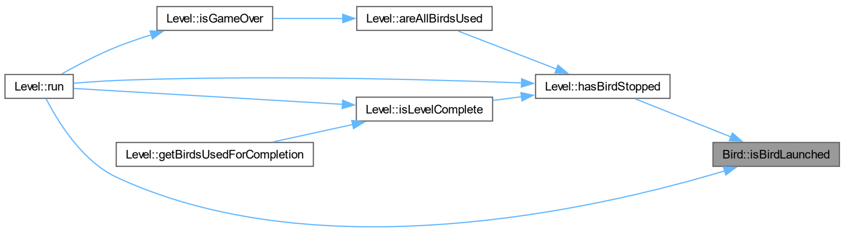
```
bool Bird::isBirdLaunched ( ) const
```

Check if the bird has been launched.

Returns

True if the bird has been launched, otherwise false.

Here is the caller graph for this function:



7.1.3.6 launch()

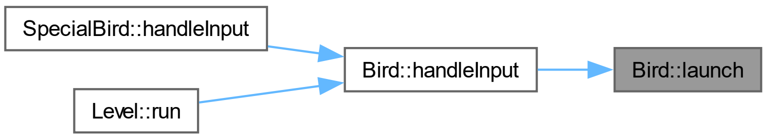
```
void Bird::launch (
            const b2Vec2 & force )
```

Launches the bird with a specified force.

Parameters

force	The force vector to apply for the launch.

Here is the caller graph for this function:



7.1.3.7 render()

```
void Bird::render (
             sf::RenderWindow & window )
```

Renders the bird on the provided SFML window.

Parameters

window	The SFML window to render on.

7.1.3.8 update()

```
void Bird::update ( )
```

Updates the bird's state.

Updates the bird's position and rotation based on its Box2D body.

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/bird.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/bird.cpp

7.2 Box Class Reference

Represents a box object in the game.

```
#include <box.hpp>
```

Public Member Functions

- Box (b2World ∗world, const sf::Texture &texture, const b2Vec2 &position)

    Constructor for the Box class.
- void update ()

    Updates the box's state.
- void render (sf::RenderWindow &window)

    Renders the box on the provided SFML window.
- b2Body ∗ getBody () const

    Get the Box2D body of the box.
- b2Vec2 getPosition () const

    Get the position of the box's Box2D body.

7.2.1 Detailed Description

Represents a box object in the game.

This class manages the rendering and physics of a box.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Box()

```
Box::Box (
            b2World ∗ world,
            const sf::Texture & texture,
            const b2Vec2 & position )
```

Constructor for the Box class.

Constructs a Box object.

Parameters

world	Pointer to the Box2D world.
texture	The texture for the box.
position	The initial position of the box.

7.2.3 Member Function Documentation

7.2.3.1 getBody()

```
b2Body ∗ Box::getBody ( ) const
```

Get the Box2D body of the box.

Returns

A pointer to the Box2D body of the box.

7.2.3.2 getPosition()

```
b2Vec2 Box::getPosition ( ) const
```

Get the position of the box's Box2D body.

Returns

The position vector of the box's body.

7.2.3.3 render()

```
void Box::render (
             sf::RenderWindow & window )
```

Renders the box on the provided SFML window.

Parameters

window	The SFML window to render on.

7.2.3.4 update()

```
void Box::update ( )
```

Updates the box's state.

Updates the box's position and rotation based on its Box2D body.

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

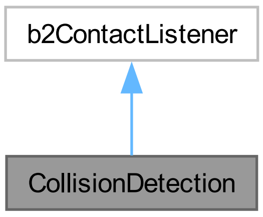
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/box.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/box.cpp

7.3 CollisionDetection Class Reference

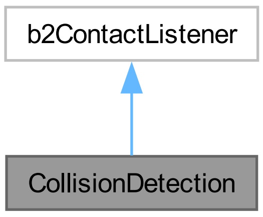
Handles collision detection and contact events.

```
#include <collisiondetection.hpp>
```

Inheritance diagram for CollisionDetection:



Collaboration diagram for CollisionDetection:



Public Member Functions

- void BeginContact (b2Contact ∗contact) override

    Called when two Box2D fixtures begin contact.
- bool isBirdFixture (b2Fixture ∗fixture)

    Checks if the fixture belongs to a bird.

7.3.1 Detailed Description

Handles collision detection and contact events.

This class extends b2ContactListener to manage collision detection and contact events between Box2D fixtures, specifically handling interactions between pigs and birds.

7.3.2 Member Function Documentation

7.3.2.1 BeginContact()

```
void CollisionDetection::BeginContact (
             b2Contact ∗ contact )  [override]
```

Called when two Box2D fixtures begin contact.

Handles the beginning of contact between two Box2D fixtures.

Parameters

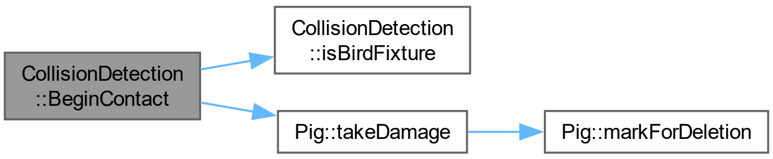
contact	A pointer to the contact object.

This function identifies the fixtures involved in the contact and determines the type of entities (pig or bird) interacting, adjusting their health accordingly.

Parameters

contact	A pointer to the contact object.

Here is the call graph for this function:


CollisionDetection ::BeginContact
CollisionDetection ::isBirdFixture
Pig::takeDamage
Pig::markForDeletion

7.3.2.2 isBirdFixture()

```
bool CollisionDetection::isBirdFixture (
            b2Fixture * fixture )
```

Checks if the fixture belongs to a bird.

Checks if the given fixture belongs to a bird.

Parameters

fixture	A pointer to the Box2D fixture to check.

Returns

True if the fixture belongs to a bird, otherwise false.

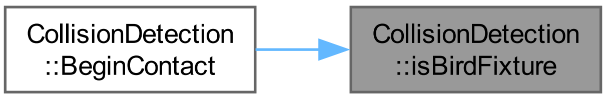
Parameters

fixture	A pointer to the Box2D fixture.

Returns

True if the fixture belongs to a bird, otherwise false.

Here is the caller graph for this function:


CollisionDetection ::BeginContact
CollisionDetection ::isBirdFixture

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/collisiondetection.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/collisiondetection.cpp

7.4 Game Class Reference

Represents the main game control.

```
#include <game.hpp>
```

Public Member Functions

- **Game** ()

  Constructs a Game object.
- void **run** ()

  Runs the game loop.

7.4.1 Detailed Description

Represents the main game control.

The Game class manages the overall control flow and execution of the game. It initializes the game components, such as the graphical user interface (GUI), and runs the main game loop.

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/game.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/game.cpp

7.5 GUI Class Reference

Manages the game's graphical user interface.

```
#include <GUI.hpp>
```

Public Member Functions

- **GUI** ()

  Constructs a GUI object and initializes the game window.
- void **run** ()

  Runs the game loop.

Public Attributes

- bool **isSpecialBird**
- int **selectedBackground**

7.5.1 Detailed Description

Manages the game's graphical user interface.

The GUI class handles the rendering and interaction components of the game's user interface. It manages various textures, fonts, window settings, buttons, game state variables, and screens. This class processes user input, updates GUI states, and renders GUI components for different screens.

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/GUI.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/GUI.cpp

7.6 Level Class Reference

Represents a game level with physics simulation.

`#include <level.hpp>`

Public Member Functions

- Level (sf::RenderWindow &win, int number, const sf::Texture &backTex, const std::string &levelFile, bool is←
  SpecialBird, bool noGravity)

  Constructs a Level object.
- ∼**Level** ()

  Destroys the Level object.
- void **run** ()

  Runs the game level.
- void **setupWorld** ()

  Sets up the Box2D world and initializes objects.
- void **createFloor** ()

  Creates the floor of the level.
- void **createBoundaries** ()

  Creates boundaries around the level.
- void createBoundary (float x, float y, float width, float height)

  Creates a boundary in the world.

- void loadObjects (const std::string &levelFile)

  Loads game objects from a level file.
- void initializeBirds (const sf::Texture &birdTex, bool isSpecialBird)

  Initializes bird objects.
- void nextBird (const sf::Texture &birdTex, bool isSpecialBird)

  Moves to the next bird in the sequence.
- bool hasBirdStopped () const

  Checks if the current bird has stopped moving.
- bool isLevelComplete () const

  Checks if the level is complete.
- bool isGameOver () const

  Checks if the game is over.
- bool areAllPigsDestroyed () const

  Checks if all pigs have been destroyed.
- bool areAllBirdsUsed () const

  Checks if all birds have been used.
- int getBirdsUsedForCompletion ()

  Gets the number of birds used to complete the level.
- Bird ∗ **getCurrentBird** ()

Public Attributes

- bool **isSpecialBird**

7.6.1 Detailed Description

Represents a game level with physics simulation.

This class manages the elements and behavior of a game level, including Box2D physics simulation, game objects like birds, pigs, boxes, and walls, as well as handling the game's progression and state.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Level()

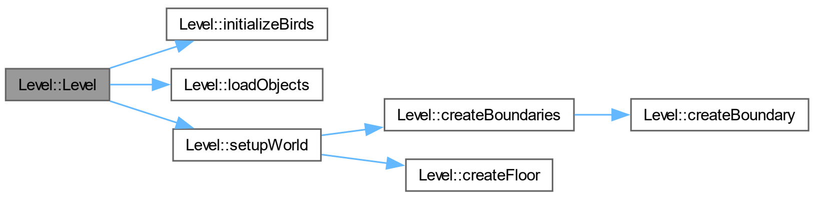
```
Level::Level (
             sf::RenderWindow & win,
             int number,
             const sf::Texture & backTex,
             const std::string & levelFile,
             bool isSpecialBirdParam,
             bool noGravity )
```

Constructs a Level object.

Parameters

win	Reference to the game's window.
number	The level number.
birdTex	The texture for the bird.
backTex	The texture for the background.
levelFile	The path to the level file.

Here is the call graph for this function:



7.6.3 Member Function Documentation

7.6.3.1 areAllBirdsUsed()

```
bool Level::areAllBirdsUsed ( ) const
```

Checks if all birds have been used.

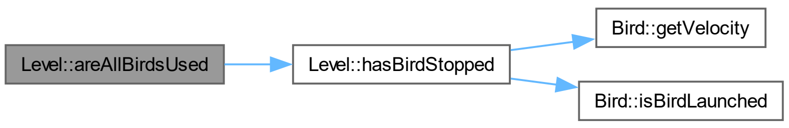
Checks if all birds in the level have been used and stopped.

Returns

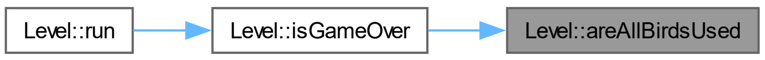
True if all birds are used, otherwise false.

True if all birds are used and stopped, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.2 areAllPigsDestroyed()

```
bool Level::areAllPigsDestroyed ( ) const
```

Checks if all pigs have been destroyed.

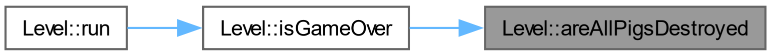
Checks if all pigs in the level have been destroyed.

Returns

True if all pigs are destroyed, otherwise false.

True if all pigs are destroyed, false otherwise.

Here is the caller graph for this function:



7.6.3.3 createBoundary()

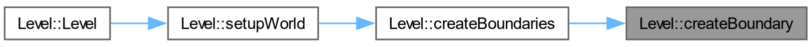
```
void Level::createBoundary (
            float x,
            float y,
            float width,
            float height )
```

Creates a boundary in the world.

Parameters

x	The x-coordinate of the boundary.
y	The y-coordinate of the boundary.
width	The width of the boundary.
height	The height of the boundary.

Here is the caller graph for this function:



7.6.3.4 getBirdsUsedForCompletion()

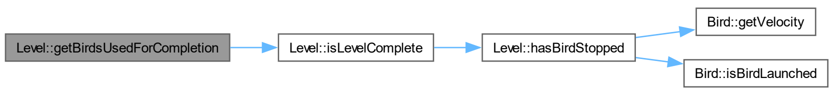
```
int Level::getBirdsUsedForCompletion ( )
```

Gets the number of birds used to complete the level.

Returns

The number of birds used if the level is completed, otherwise -1.

Here is the call graph for this function:



7.6.3.5 hasBirdStopped()

```
bool Level::hasBirdStopped ( ) const
```

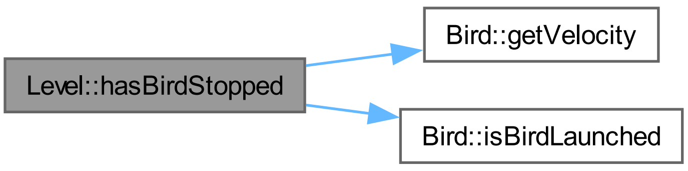
Checks if the current bird has stopped moving.

Returns

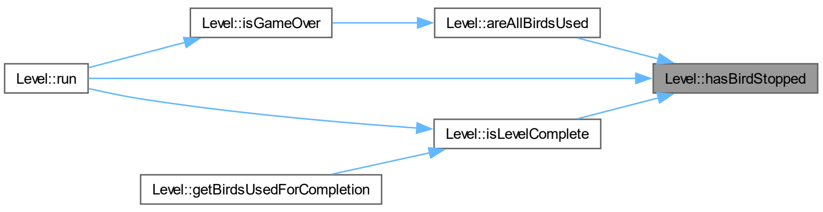
True if the bird has stopped, otherwise false.

True if the bird has stopped, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.6 initializeBirds()

```
void Level::initializeBirds (
             const sf::Texture & birdTex,
             bool isSpecialBird )
```

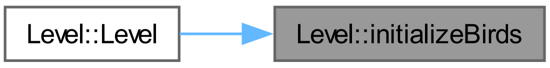
Initializes bird objects.

Initializes the bird objects for the level.

Parameters

birdTex	The texture for birds.
birdTex	The texture for the birds.

Here is the caller graph for this function:



7.6.3.7 isGameOver()

```
bool Level::isGameOver ( ) const
```

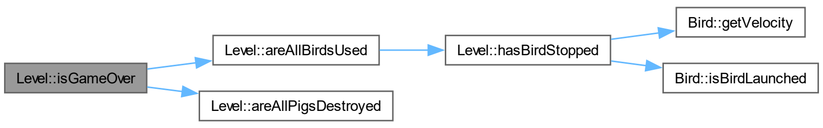
Checks if the game is over.

Returns

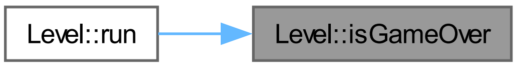
True if the game is over, otherwise false.

True if the game is over, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.8 isLevelComplete()

```
bool Level::isLevelComplete ( ) const
```

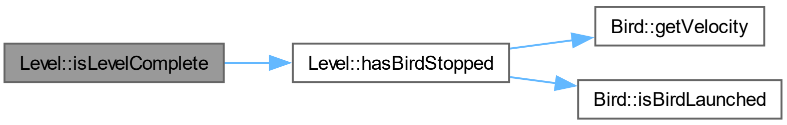
Checks if the level is complete.

Returns

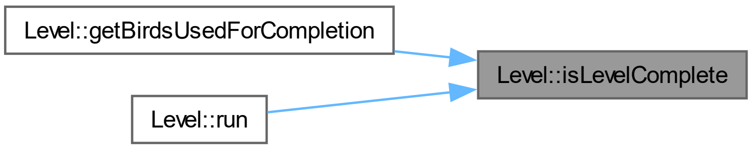
True if the level is complete, otherwise false.

True if the level is complete, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.9 loadObjects()

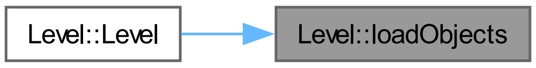
```
void Level::loadObjects (
            const std::string & levelFile )
```

Loads game objects from a level file.

Parameters

levelFile	The path to the level file.

Here is the caller graph for this function:



7.6.3.10 nextBird()

```
void Level::nextBird (
            const sf::Texture & birdTex,
            bool isSpecialBird )
```

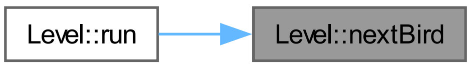
Moves to the next bird in the sequence.

Moves to the next bird in the level.

Parameters

birdTex	The texture for birds.
birdTex	The texture for the birds.

Here is the caller graph for this function:



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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/level.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/level.cpp

7.7 LevelEditor Class Reference

Represents a tool to create game levels and edit them visually.

```
#include <leveleditor.hpp>
```

Public Member Functions

- LevelEditor (sf::RenderWindow &win, int number, const sf::Texture &backTex)

  Constructs a Level Editor object.
- std::string **run** ()

  Runs the events of the Level Editor.
- void **setUpLevel** ()

  Sets up the Level Editor with objects.
- void addObject (int chosenObject, sf::Vector2f mousePos, std::string &filePath)

  Adds objects to CSV file.
- void **drawObject** ()

  Draws the objects on the screen.

7.7.1 Detailed Description

Represents a tool to create game levels and edit them visually.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 LevelEditor()

```
LevelEditor::LevelEditor (
             sf::RenderWindow & win,
             int number,
             const sf::Texture & backTex )
```

Constructs a Level Editor object.

Parameters

win	Reference to the game's window.
number	The background number for the level.
backTex	The texture for the background.

7.7.3 Member Function Documentation

7.7.3.1 addObject()

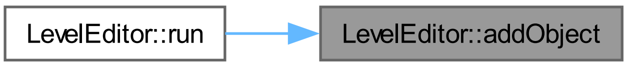
```
void LevelEditor::addObject (
             int chosenObject,
             sf::Vector2f mousePos,
             std::string & filePath )
```

Adds objects to CSV file.

Parameters

chosenObject	The object added to the game.
mousePos	The position of the added object.
filePath	The path to the CSV file to add to.

Here is the caller graph for this function:



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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/leveleditor.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/leveleditor.cpp

7.8 ObjectData Class Reference

Reads a CSV file that makes up the game levels.

```
#include <readfile.hpp>
```

Public Attributes

- std::string **type**

  < Object type
- float **x**
- float **y**

7.8.1 Detailed Description

Reads a CSV file that makes up the game levels.

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/readfile.hpp

7.9 Pig Class Reference

Represents a pig object in the game.

```
#include <pig.hpp>
```

Public Member Functions

- Pig (b2World ∗world, const sf::Texture &texture, const b2Vec2 &position)

    Constructs a Pig object.
- void update ()

    Updates the pig's state.
- void render (sf::RenderWindow &window)

    Renders the pig on the given window.
- void takeDamage (float damage)

    Applies damage to the pig's health.
- bool alive () const

    Checks if the pig is alive.
- void destroyBody ()

    Destroys the Box2D body associated with the pig.
- void **markForDeletion** ()

    Marks the pig for deletion.
- bool isMarkedForDeletion () const

    Checks if the pig is marked for deletion.
- float getHealth () const

    Retrieves the pig's health.
- b2Body ∗ getBody () const

    Retrieves the Box2D body associated with the pig.

7.9.1 Detailed Description

Represents a pig object in the game.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Pig()

```
Pig::Pig (
            b2World * world,
            const sf::Texture & texture,
            const b2Vec2 & position )
```

Constructs a Pig object.

Parameters

world	Pointer to the Box2D world.
texture	Texture for the pig.
position	Initial position of the pig.
world	Pointer to the Box2D world.
texture	The texture for the pig.
position	The initial position of the pig.

7.9.3 Member Function Documentation

7.9.3.1 alive()

```
bool Pig::alive ( ) const
```

Checks if the pig is alive.

Returns

true if the pig is alive, false otherwise.

7.9.3.2 destroyBody()

```
void Pig::destroyBody ( )
```

Destroys the Box2D body associated with the pig.

Destroys the Box2D body associated with the pig if it exists and the world is not locked.

7.9.3.3 getBody()

```
b2Body * Pig::getBody ( ) const
```

Retrieves the Box2D body associated with the pig.

Returns

Pointer to the pig's Box2D body.

7.9.3.4 getHealth()

```
float Pig::getHealth ( ) const
```

Retrieves the pig's health.

Returns

The current health of the pig.

7.9.3.5 isMarkedForDeletion()

```
bool Pig::isMarkedForDeletion ( ) const
```

Checks if the pig is marked for deletion.

Returns

true if marked for deletion, false otherwise.

7.9.3.6 render()

```
void Pig::render (
             sf::RenderWindow & window )
```

Renders the pig on the given window.

Renders the pig on the given window if it is alive.

Parameters

window	The SFML render window.

7.9.3.7 takeDamage()

```
void Pig::takeDamage (
             float damage )
```

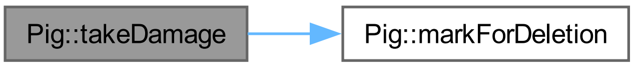
Applies damage to the pig's health.

Applies damage to the pig's health and marks it for deletion if health reaches zero or below.

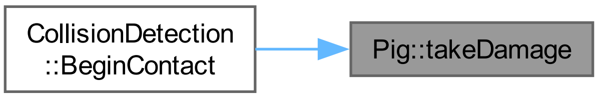
Parameters

damage	Amount of damage to apply.

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.3.8 update()

```
void Pig::update ( )
```

Updates the pig's state.

Updates the pig's position and rotation if it is alive.

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/pig.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/pig.cpp

7.10 Player Class Reference

Represents a player in the game, storing their name and level scores.

`#include <player.hpp>`

Public Member Functions

- Player (std::string name)
  Constructs a Player object.
- std::string loadFromFile ()
  Loads player data from a file.
- void **saveToFile** ()
  Saves player data to a file.
- int getScoreForLevel (int levelNumber)
  Gets the score earned by the player for a specific level.
- void updateScore (int levelNumber, int stars)
  Updates the score for a specific level if it's higher than the previous score.

Public Attributes

- std::string **name**
- std::map< int, int > **levelScores**

7.10.1 Detailed Description

Represents a player in the game, storing their name and level scores.

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

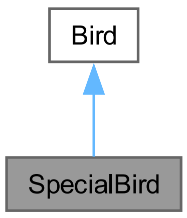
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/player.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/player.cpp

7.11 SpecialBird Class Reference

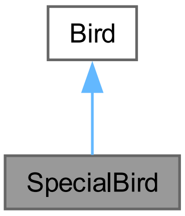
Represents a special type of bird with specific shooting behavior.

`#include <specialbird.hpp>`

Inheritance diagram for SpecialBird:



Collaboration diagram for SpecialBird:



Public Member Functions

- SpecialBird (b2World $*$world, const sf::Texture &texture, const b2Vec2 &position)

  Constructs a SpecialBird object.
- void handleInput (const sf::Event &event, const sf::RenderWindow &window) override

  Handles input events specific to the SpecialBird.
- bool getShot ()

  Checks if the SpecialBird has been shot.

Public Member Functions inherited from Bird

- Bird (b2World $*$world, const sf::Texture &texture, const b2Vec2 &position)

  Constructor for the Bird class.
- void update ()

  Updates the bird's state.
- void render (sf::RenderWindow &window)

  Renders the bird on the provided SFML window.
- void launch (const b2Vec2 &force)

  Launches the bird with a specified force.
- b2Vec2 getVelocity () const

  Get the current velocity of the bird.
- bool isBirdLaunched () const

  Check if the bird has been launched.
- b2Body $*$ getBody () const

  Get the Box2D body of the bird.
- std::vector$<$ sf::CircleShape $>$ calculateTrajectory (const sf::Vector2f &launchVector, int numDots)

  Calculate the trajectory points.

Additional Inherited Members

Public Attributes inherited from Bird

- b2Body $*$ **body**

  The bird's Box2D body.
- std::vector$<$ sf::CircleShape $>$ **trajectoryDots**

  Stores the trajectory dots.

7.11.1 Detailed Description

Represents a special type of bird with specific shooting behavior.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 SpecialBird()

```
SpecialBird::SpecialBird (
            b2World * world,
            const sf::Texture & texture,
            const b2Vec2 & position )
```

Constructs a SpecialBird object.

Parameters

world	Pointer to the Box2D world.
texture	The texture for the special bird.
position	The initial position of the special bird.

7.11.3 Member Function Documentation

7.11.3.1 getShot()

```
bool SpecialBird::getShot ( )
```

Checks if the SpecialBird has been shot.

Returns

true if the SpecialBird has been shot, false otherwise.

7.11.3.2 handleInput()

```
void SpecialBird::handleInput (
            const sf::Event & event,
            const sf::RenderWindow & window )  [override], [virtual]
```

Handles input events specific to the SpecialBird.

Handles input events for the SpecialBird.

Parameters

event	The SFML event to handle.
window	The SFML render window.

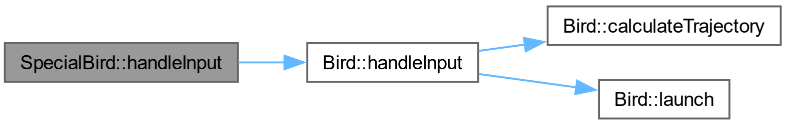
Overrides the base class function to handle specific input for the SpecialBird.

Parameters

event	The SFML event to handle.
window	The SFML render window.

Reimplemented from Bird.

Here is the call graph for this function:



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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/specialbird.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/specialbird.cpp

7.12 Wall Class Reference

Represents a wall object in the game.

```
#include <wall.hpp>
```

Public Member Functions

- Wall (b2World ∗world, const sf::Texture &texture, const b2Vec2 &position)

    Constructs a wall object.
- void **update** ()

    Updates the wall's position and state.
- void render (sf::RenderWindow &window)

    Renders the wall on the specified window.
- b2Body ∗ getBody () const

    Gets the Box2D body associated with the wall.
- b2Vec2 getPosition () const

    Gets the position of the wall.

7.12.1 Detailed Description

Represents a wall object in the game.

This class defines a wall object that can be used in the game. It includes methods for updating and rendering the wall.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Wall()

```
Wall::Wall (
            b2World * world,
            const sf::Texture & texture,
            const b2Vec2 & position )
```

Constructs a wall object.

Parameters

world	A pointer to the Box2D world.
texture	The texture to be used for the wall sprite.
position	The initial position of the wall.

< Adjust the size of the wall

< About 2 times smaller than wallShape

7.12.3 Member Function Documentation

7.12.3.1 getBody()

```
b2Body * Wall::getBody ( ) const
```

Gets the Box2D body associated with the wall.

Returns

A pointer to the Box2D body.

7.12.3.2 getPosition()

```
b2Vec2 Wall::getPosition ( ) const
```

Gets the position of the wall.

Returns

The position of the wall in the Box2D world.

7.12.3.3 render()

```
void Wall::render (
            sf::RenderWindow & window )
```

Renders the wall on the specified window.

Parameters

window	The rendering window to draw the wall on.

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

- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/wall.hpp
- //wsl.localhost/Ubuntu/home/caspertillander/cpp-course-autumn-2023/Project/src/wall.cpp

Chapter 8

File Documentation

8.1 bird.hpp

```
#ifndef BIRD_HPP
#define BIRD_HPP

#include <SFML/Graphics.hpp>
#include <box2d/box2d.h>
#include <vector>

class Bird {
private:
    // Textures used in class
    sf::CircleShape birdShape;

    // State variables
    bool isLaunched;
    bool isDragging;
    sf::Vector2f initialClickPosition;

    // Constants
    const float FORCE_MULTIPLIER = 200.0f;
    int currentBirdIndex = 0;
    int totalBirds = 3;

public:
    b2Body* body;

    Bird(b2World* world, const sf::Texture& texture, const b2Vec2& position);

    void update();

    void render(sf::RenderWindow& window);

    virtual void handleInput(const sf::Event& event, const sf::RenderWindow& window);

    void launch(const b2Vec2& force);

    b2Vec2 getVelocity() const;

    bool isBirdLaunched() const;

    b2Body* getBody() const;

    std::vector<sf::CircleShape> calculateTrajectory(const sf::Vector2f& launchVector, int numDots);
    std::vector<sf::CircleShape> trajectoryDots;
};

#endif // BIRD_HPP
```

8.2 box.hpp

```
#ifndef BOX_HPP
#define BOX_HPP

#include <SFML/Graphics.hpp>
#include <box2d/box2d.h>
```

```

class Box {
private:
    // Rendering and physics variables
    sf::RectangleShape boxShape;
    b2Body* body;

public:
    Box(b2World* world, const sf::Texture& texture, const b2Vec2& position);

    void update();

    void render(sf::RenderWindow& window);

    b2Body* getBody() const;
    b2Vec2 getPosition() const;
};

#endif // BOX_HPP
```

8.3 collisiondetection.hpp

```
#ifndef COLLISIONDETECTION_HPP
#define COLLISIONDETECTION_HPP

#include <box2d/box2d.h>
#include "pig.hpp"
#include "bird.hpp"

class CollisionDetection : public b2ContactListener {
public:
    void BeginContact(b2Contact* contact) override;

    bool isBirdFixture(b2Fixture* fixture);
};

#endif // COLLISIONDETECTION_HPP


```

8.4 game.hpp

```
#ifndef GAME_HPP
#define GAME_HPP

#include "GUI.hpp"

class Game {

public:
    Game();

    void run();

private:
    GUI gui;
};

#endif // GAME_HPP
```

8.5 GUI.hpp

```
#ifndef GUI_HPP
#define GUI_HPP

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include "level.hpp"
#include "bird.hpp"
#include "leveleditor.hpp"
#include "player.hpp"

class GUI {
public:
    GUI(); // Constructor
    void run(); // Runs the game's GUI

    // Public members
    bool isSpecialBird;
    int selectedBackground;

private:
    // Private members
    // Textures
    sf::Texture mainScreenTexture;
    sf::Texture backgroundTexture;
    sf::Texture soundTexture;
    sf::Texture levelBackgroundTexture;
    sf::Texture settingsBackgroundTexture;
    sf::Texture editorBackgroundTexture;
    sf::Texture gameOverBackgroundTexture;
    sf::Texture levelCompleteBackgroundTexture;
    sf::Texture chooseABirdBackgroundTexture;
    sf::Texture birdTexture;
    sf::Texture specialBirdTexture;
    sf::Texture levelEditor1;
    sf::Texture levelEditor2;
    sf::Texture levelEditor3;
    sf::Texture starTexture;

    // Fonts
    sf::Font font;

    // RenderWindow
    sf::RenderWindow window;

    // Music
    sf::Music music;

    // Texts
    sf::Text titleText;
    sf::Text playText;
    sf::Text settingsText;
    sf::Text returnToHomeText;
    sf::Text levelsText;
    sf::Text level1Text;
    sf::Text level2Text;
    sf::Text level3Text;
    sf::Text tryAgainText;
    sf::Text returnToLevelsText;
    sf::Text createLevelText;
    sf::Text gravityText;
    sf::Text playerMessage;
    sf::Text chooseBirdText;
    sf::Text levelEditorText;
    sf::Text playerNameLabel;
    sf::Text submitButtonText;
    sf::Text inputText;

    // Shapes and Buttons
    sf::CircleShape ButtonShape;
    sf::CircleShape highlightCircle;
    sf::CircleShape circleButton;
    sf::RectangleShape highlightRectangle;
    sf::RectangleShape redLine;
    sf::RectangleShape redLine2;
    sf::RectangleShape playerNameInputBox;
    sf::RectangleShape submitButton;

    // Sprites
    sf::Sprite backgroundSprite;
    sf::Sprite soundButton;
    sf::Sprite normalBirdButton;
    sf::Sprite specialBirdButton;
    sf::Sprite starSprite;
    sf::Sprite levelStarSprite;
    sf::Sprite level1Button;
    sf::Sprite level2Button;
    sf::Sprite level3Button;

    // Game State Variables
    Level* currentLevel;
    enum Screen { Home, BirdSelection, Levels, PlayingLevel, GameOver, LevelCompleted, Settings,
LevelEditorSelection, PlayingLevelEditor };
    Screen currentScreen;
    int levelNumberEditor;
    LevelEditor* currentLevelEditor;
    std::string pathToCreatedFile;
    Player* currentPlayer;
    std::string playerNameInput;

    // Additional int and bool members
    int levelNumber;
    sf::Vector2u textureSize;
    sf::Vector2u windowSize;
    bool soundOn;
    bool noGravity;
    bool isLevelEditorLevel;

    // Private member functions
    void initialize();
    void processEvents();
    void update();
    void render();
    void startGame();
    void drawHomeScreen();
    void drawLevelsScreen();
    void launchLevel(int levelNumber);
    void drawGameOverScreen();
    void drawLevelCompletedScreen();
    void updateButtonHoverEffect(sf::Text& buttonText, sf::Vector2f mousePos);
    void drawSettingsScreen();
    void setupButton(sf::Text& buttonText, const std::string& text);
    void updateBackground();
    void drawBirdSelectionScreen();
    void drawLevelEditorSelectionScreen();
    void launchLevelEditor(int levelNumberEditor);
    void launchLevelEditorLevel(int levelNumberEditor, std::string filePath);
};

#endif // GUI_HPP
```

8.6 level.hpp

```
#ifndef LEVEL_HPP
#define LEVEL_HPP

#include <SFML/Graphics.hpp>
#include <box2d/box2d.h>
#include <vector>
#include "bird.hpp"
#include "specialbird.hpp"
#include "pig.hpp"
#include "box.hpp"
#include "wall.hpp"
#include "readfile.hpp"
#include "collisiondetection.hpp"

class Level {
private:
    // Level properties
    int levelNumber;
    int currentBirdIndex = 0;
    int totalBirds = 3;
    sf::RenderWindow& window;
    sf::Texture backgroundTexture;
    sf::Sprite backgroundSprite;
    b2World* world;
    bool noGravity;

    // Level objects
    std::vector<Pig*> pigs;
    std::vector<Box*> boxes;
    std::vector<Wall*> walls;
    std::vector<Bird*> birds;

    //Textures used in class
    sf::Texture pigTexture;
    sf::Texture boxTexture;
    sf::Texture wallTexture;
    sf::Texture birdTexture;
    sf::Font font;

    // UI elements
    sf::Text birdsRemainingText;
    sf::Text pigsRemainingText;

public:
    Level(sf::RenderWindow& win, int number, const sf::Texture& backTex, const std::string& levelFile,
bool isSpecialBird, bool noGravity);


    bool isSpecialBird;
    ~Level();

    void run();

    void setupWorld();

    void createFloor();

    void createBoundaries();

    void createBoundary(float x, float y, float width, float height);

    void loadObjects(const std::string& levelFile);

    void initializeBirds(const sf::Texture& birdTex, bool isSpecialBird);

    void nextBird(const sf::Texture& birdTex, bool isSpecialBird);

    bool hasBirdStopped() const;

    bool isLevelComplete() const;

    bool isGameOver() const;

    bool areAllPigsDestroyed() const;

    bool areAllBirdsUsed() const;

    int getBirdsUsedForCompletion();

    Bird* getCurrentBird();

};

#endif // LEVEL_HPP





```

8.7 leveleditor.hpp

```
#ifndef LEVEL_EDITOR_HPP
#define LEVEL_EDITOR_HPP

#include <SFML/Graphics.hpp>
#include <box2d/box2d.h>

class LevelEditor {
private:
// Member variables
    int levelNumber;
    int chosenObject = 1;

    sf::Text playCreatedLevelText;
    sf::Font font;
    sf::CircleShape ButtonShape;

    sf::RenderWindow& window;
    sf::Texture backgroundTexture;
    sf::Sprite backgroundSprite;

    sf::Texture pigTexture;
    sf::Texture wallTexture;
    sf::Texture boxTexture;

    sf::Sprite pigSprite;
    sf::Sprite wallSprite;
    sf::Sprite boxSprite;

    sf::CircleShape pigShape;
    sf::RectangleShape boxShape;
    sf::RectangleShape wallShape;

    std::vector<sf::Vector2f> pigPositions;
    std::vector<sf::Vector2f> boxPositions;
    std::vector<sf::Vector2f> wallPositions;

    sf::RectangleShape highlightRectangle;

    std::string filePath = "../Createdlevels/createdlevel.csv";

public:
    LevelEditor(sf::RenderWindow& win, int number, const sf::Texture& backTex);

    std::string run();

    void setUpLevel();

    void addObject(int chosenObject, sf::Vector2f mousePos, std::string& filePath);

    void drawObject();

};


#endif // LEVEL_EDITOR_HPP
```

8.8 pig.hpp

```
#ifndef PIG_HPP
#define PIG_HPP

#include <SFML/Graphics.hpp>
#include <box2d/box2d.h>

class Pig {
private:
    // Rendering and physics variables
    sf::CircleShape pigShape;
    b2Body* body;

    // State variables
    float health;
    bool isAlive;
    bool markedForDeletion;

public:
    Pig(b2World* world, const sf::Texture& texture, const b2Vec2& position);

    void update();

    void render(sf::RenderWindow& window);

    void takeDamage(float damage);

    bool alive() const;

    void destroyBody();

    void markForDeletion();

    bool isMarkedForDeletion() const;

    float getHealth() const;

    b2Body* getBody() const;
};

#endif // PIG_HPP
```

8.9 player.hpp

```
#ifndef PLAYER_HPP
#define PLAYER_HPP

#include <string>
#include <map>

class Player {
public:
    // Player name
    std::string name;

    std::map<int, int> levelScores;

    Player(std::string name);

    std::string loadFromFile();

    void saveToFile();

    int getScoreForLevel(int levelNumber);
    void updateScore(int levelNumber, int stars);
};

#endif // PLAYER_HPP
```

8.10 readfile.hpp

```
#ifndef READFILE_HPP
#define READFILE_HPP

#include <string>
#include <vector>

struct ObjectData {
    std::string type;

    // Coordinates
    float x, y;
};

std::vector<ObjectData> readLevelData(const std::string& filename);

#endif // READFILE_HPP

```

8.11 specialbird.hpp

```
#ifndef SPECIALBIRD_HPP
#define SPECIALBIRD_HPP

#include "bird.hpp"

class SpecialBird : public Bird {
public:
    SpecialBird(b2World* world, const sf::Texture& texture, const b2Vec2& position);

    void handleInput(const sf::Event& event, const sf::RenderWindow& window) override;

    bool getShot();

private:
    void shootTowardsClick(const sf::Vector2f& targetPosition);

    bool isShot;
};

#endif // SPECIALBIRD_HPP
```

8.12 wall.hpp

```
#ifndef WALL_HPP
#define WALL_HPP

#include <SFML/Graphics.hpp>
#include <box2d/box2d.h>

class Wall {
private:
    sf::RectangleShape wallShape;
    b2Body* body;

public:
    Wall(b2World* world, const sf::Texture& texture, const b2Vec2& position);

    void update();

    void render(sf::RenderWindow& window);

    b2Body* getBody() const;


    b2Vec2 getPosition() const;
};

#endif // WALL_HPP

```

Index