

Happy Birds

Generated by Doxygen 1.9.8



<b>1 Source content</b>	<b>1</b>
1.0.1 Game	1
1.0.2 GUI	1
1.0.3 Level	1
1.0.4 ReadFile	1
1.0.5 Player	1
1.0.6 LevelEditor	1
1.0.7 CollisionDetection	2
1.0.8 Bird	2
1.0.9 SpecialBird	2
1.0.10 Pig	2
1.0.11 Box	2
1.0.12 Wall	2
<b>2 Topic Index</b>	<b>3</b>
2.1 Topics	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Topic Documentation</b>	<b>11</b>
6.1 Level Properties	11
6.1.1 Detailed Description	11
6.1.2 Function Documentation	11
6.1.2.1 getScoreForLevel()	11
6.1.2.2 loadFromFile()	12
6.1.2.3 Player()	12
6.1.2.4 updateScore()	12
6.2 SpecialBird	13
6.2.1 Detailed Description	13
<b>7 Class Documentation</b>	<b>15</b>
7.1 Bird Class Reference	15
7.1.1 Detailed Description	16
7.1.2 Constructor & Destructor Documentation	16
7.1.2.1 Bird()	16
7.1.3 Member Function Documentation	17
7.1.3.1 calculateTrajectory()	17

7.1.3.2	<a href="#">getBody()</a>	17
7.1.3.3	<a href="#">getVelocity()</a>	17
7.1.3.4	<a href="#">handleInput()</a>	18
7.1.3.5	<a href="#">isBirdLaunched()</a>	19
7.1.3.6	<a href="#">launch()</a>	19
7.1.3.7	<a href="#">render()</a>	20
7.1.3.8	<a href="#">update()</a>	20
7.2	<a href="#">Box Class Reference</a>	20
7.2.1	<a href="#">Detailed Description</a>	21
7.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	21
7.2.2.1	<a href="#">Box()</a>	21
7.2.3	<a href="#">Member Function Documentation</a>	21
7.2.3.1	<a href="#">getBody()</a>	21
7.2.3.2	<a href="#">getPosition()</a>	22
7.2.3.3	<a href="#">render()</a>	22
7.2.3.4	<a href="#">update()</a>	22
7.3	<a href="#">CollisionDetection Class Reference</a>	22
7.3.1	<a href="#">Detailed Description</a>	23
7.3.2	<a href="#">Member Function Documentation</a>	23
7.3.2.1	<a href="#">BeginContact()</a>	23
7.3.2.2	<a href="#">isBirdFixture()</a>	24
7.4	<a href="#">Game Class Reference</a>	25
7.4.1	<a href="#">Detailed Description</a>	25
7.5	<a href="#">GUI Class Reference</a>	25
7.5.1	<a href="#">Detailed Description</a>	26
7.6	<a href="#">Level Class Reference</a>	26
7.6.1	<a href="#">Detailed Description</a>	27
7.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	27
7.6.2.1	<a href="#">Level()</a>	27
7.6.3	<a href="#">Member Function Documentation</a>	28
7.6.3.1	<a href="#">areAllBirdsUsed()</a>	28
7.6.3.2	<a href="#">areAllPigsDestroyed()</a>	29
7.6.3.3	<a href="#">createBoundary()</a>	29
7.6.3.4	<a href="#">getBirdsUsedForCompletion()</a>	30
7.6.3.5	<a href="#">hasBirdStopped()</a>	30
7.6.3.6	<a href="#">initializeBirds()</a>	31
7.6.3.7	<a href="#">isGameOver()</a>	31
7.6.3.8	<a href="#">isLevelComplete()</a>	32
7.6.3.9	<a href="#">loadObjects()</a>	32
7.6.3.10	<a href="#">nextBird()</a>	33
7.7	<a href="#">LevelEditor Class Reference</a>	34
7.7.1	<a href="#">Detailed Description</a>	34

7.7.2 Constructor & Destructor Documentation	34
7.7.2.1 LevelEditor()	34
7.7.3 Member Function Documentation	34
7.7.3.1 addObject()	34
7.8 ObjectData Class Reference	35
7.8.1 Detailed Description	35
7.9 Pig Class Reference	36
7.9.1 Detailed Description	36
7.9.2 Constructor & Destructor Documentation	36
7.9.2.1 Pig()	36
7.9.3 Member Function Documentation	37
7.9.3.1 alive()	37
7.9.3.2 destroyBody()	37
7.9.3.3 getBody()	37
7.9.3.4 getHealth()	37
7.9.3.5 isMarkedForDeletion()	37
7.9.3.6 render()	37
7.9.3.7 takeDamage()	38
7.9.3.8 update()	38
7.10 Player Class Reference	39
7.10.1 Detailed Description	39
7.11 SpecialBird Class Reference	39
7.11.1 Detailed Description	41
7.11.2 Constructor & Destructor Documentation	41
7.11.2.1 SpecialBird()	41
7.11.3 Member Function Documentation	41
7.11.3.1 getShot()	41
7.11.3.2 handleInput()	41
7.12 Wall Class Reference	42
7.12.1 Detailed Description	42
7.12.2 Constructor & Destructor Documentation	43
7.12.2.1 Wall()	43
7.12.3 Member Function Documentation	43
7.12.3.1 getBody()	43
7.12.3.2 getPosition()	43
7.12.3.3 render()	43
<b>8 File Documentation</b>	<b>45</b>
8.1 bird.hpp	45
8.2 box.hpp	45
8.3 collisiondetection.hpp	46
8.4 game.hpp	46

8.5 GUI.hpp . . . . .	46
8.6 level.hpp . . . . .	48
8.7 leveleditor.hpp . . . . .	49
8.8 pig.hpp . . . . .	50
8.9 player.hpp . . . . .	50
8.10 readfile.hpp . . . . .	51
8.11 specialbird.hpp . . . . .	51
8.12 wall.hpp . . . . .	51
<b>Index</b>	<b>53</b>

# 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:

Level Properties . . . . .	<a href="#">11</a>
SpecialBird . . . . .	<a href="#">13</a>



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

b2ContactListener	
CollisionDetection . . . . .	<a href="#">22</a>
Bird . . . . .	<a href="#">15</a>
SpecialBird . . . . .	<a href="#">39</a>
Box . . . . .	<a href="#">20</a>
Game . . . . .	<a href="#">25</a>
GUI . . . . .	<a href="#">25</a>
Level . . . . .	<a href="#">26</a>
LevelEditor . . . . .	<a href="#">34</a>
ObjectData . . . . .	<a href="#">35</a>
Pig . . . . .	<a href="#">36</a>
Player . . . . .	<a href="#">39</a>
Wall . . . . .	<a href="#">42</a>



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">Bird</a>	Represents a bird object in the game . . . . .	15
<a href="#">Box</a>	Represents a box object in the game . . . . .	20
<a href="#">CollisionDetection</a>	Handles collision detection and contact events . . . . .	22
<a href="#">Game</a>	Represents the main game control . . . . .	25
<a href="#">GUI</a>	Manages the game's graphical user interface . . . . .	25
<a href="#">Level</a>	Represents a game level with physics simulation . . . . .	26
<a href="#">LevelEditor</a>	Represents a tool to create game levels and edit them visually . . . . .	34
<a href="#">ObjectData</a>	Reads a CSV file that makes up the game levels . . . . .	35
<a href="#">Pig</a>	Represents a pig object in the game . . . . .	36
<a href="#">Player</a>	Represents a player in the game, storing their name and level scores . . . . .	39
<a href="#">SpecialBird</a>	Represents a special type of bird with specific shooting behavior . . . . .	39
<a href="#">Wall</a>	Represents a wall object in the game . . . . .	42



# Chapter 5

## File Index

### 5.1 File List

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

//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/bird.hpp . . . . .	45
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/box.hpp . . . . .	45
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/collisiondetection.hpp	46
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/game.hpp . . . . .	46
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/GUI.hpp . . . . .	46
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/level.hpp . . . . .	48
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/leveeditor.hpp . . . .	49
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/pig.hpp . . . . .	50
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/player.hpp . . . . .	50
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/readfile.hpp . . . .	51
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/specialbird.hpp . . .	51
//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/wall.hpp . . . . .	51





# 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**

<i>levelNumber</i>	The level number.
--------------------	-------------------

**Returns**

The score earned by the player for the specified level.

**Parameters**

<i>levelNumber</i>	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**

<i>name</i>	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

<i>levelNumber</i>	The number of the level.
<i>stars</i>	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

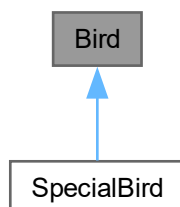
# Class Documentation

### 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

<i>world</i>	Pointer to the Box2D world.
<i>texture</i>	The texture for the bird.
<i>position</i>	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

<i>launchVector</i>	The vector representing the launch direction.
<i>numDots</i>	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.

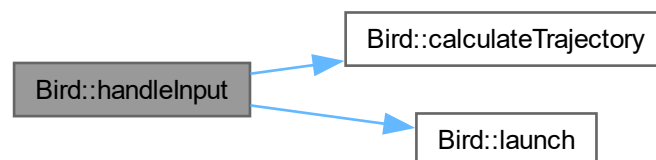
Handles input events for the bird, including dragging and launching.

**Parameters**

<i>event</i>	The SFML event to handle.
<i>window</i>	The SFML window associated with the event.

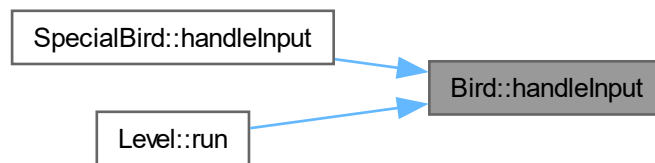
Reimplemented in [SpecialBird](#).

Here is the call graph for this function:





Here is the caller graph for this function:



### 7.1.3.5 isBirdLaunched()

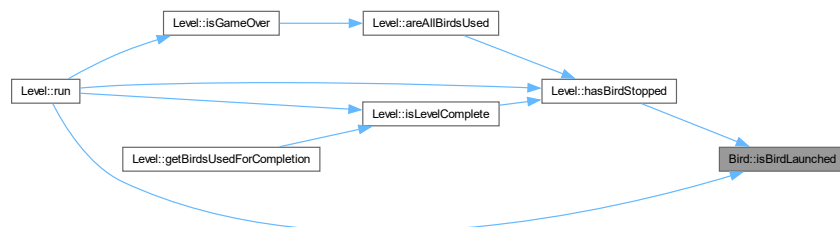
```
bool Bird::isBirdLaunched ( ) const
```

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()

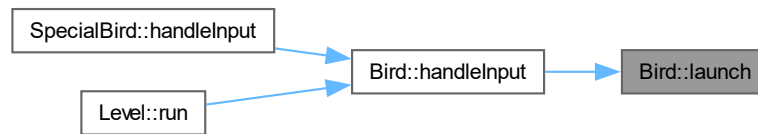
```
void Bird::launch (
    const b2Vec2 & force )
```

Launches the bird with a specified force.

#### Parameters

<i>force</i>	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

<i>window</i>	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/casptillander/cpp-course-autumn-2023/Project/src/bird.hpp
- //wsl.localhost/Ubuntu/home/casptillander/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

<i>world</i>	Pointer to the Box2D world.
<i>texture</i>	The texture for the box.
<i>position</i>	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

<i>window</i>	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:

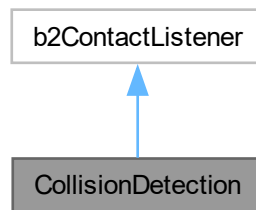
- `//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/box.hpp`
- `//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/box.cpp`

## 7.3 CollisionDetection Class Reference

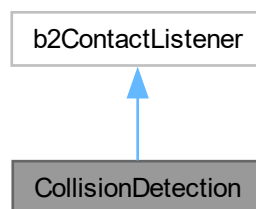
Handles collision detection and contact events.

```
#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

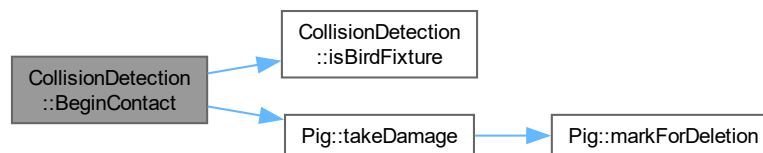
<i>contact</i>	A pointer to the contact object.
----------------	----------------------------------

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

## Parameters

<i>contact</i>	A pointer to the contact object.
----------------	----------------------------------

Here is the call graph for this function:



## 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

<i>fixture</i>	A pointer to the Box2D fixture to check.
----------------	--

## Returns

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

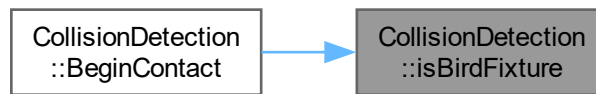
## Parameters

<i>fixture</i>	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:



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

- `//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/collisiondetection.hpp`
- `//wsl.localhost/Ubuntu/home/casptertillander/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/casptertillander/cpp-course-autumn-2023/Project/src/game.hpp`
- `//wsl.localhost/Ubuntu/home/casptertillander/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/casptertillander/cpp-course-autumn-2023/Project/src/GUI.hpp`
- `//wsl.localhost/Ubuntu/home/casptertillander/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 isSpecialBird, 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()

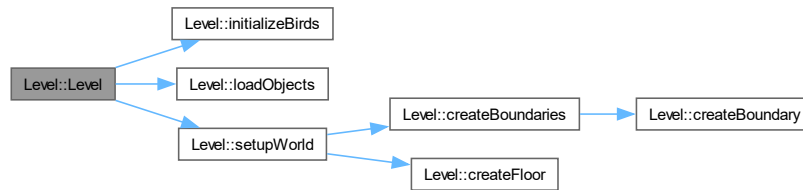
```
Level::Level (
    sf::RenderWindow & win,
    int number,
    const sf::Texture & backTex,
    const std::string & levelFile,
    bool isSpecialBirdParam,
    bool noGravity )
```

Constructs a [Level](#) object.

#### Parameters

<i>win</i>	Reference to the game's window.
<i>number</i>	The level number.
<i>birdTex</i>	The texture for the bird.
<i>backTex</i>	The texture for the background.
<i>levelFile</i>	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

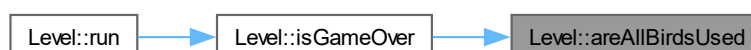
True if all birds are used, otherwise false.

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()

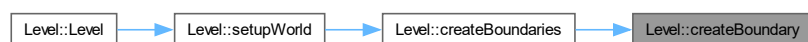
```
void Level::createBoundary (
    float x,
    float y,
    float width,
    float height )
```

Creates a boundary in the world.

#### Parameters

<i>x</i>	The x-coordinate of the boundary.
<i>y</i>	The y-coordinate of the boundary.
<i>width</i>	The width of the boundary.
<i>height</i>	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
```

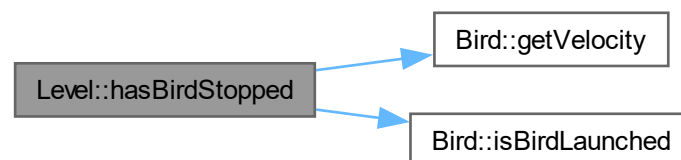
Checks if the current bird has stopped moving.

##### Returns

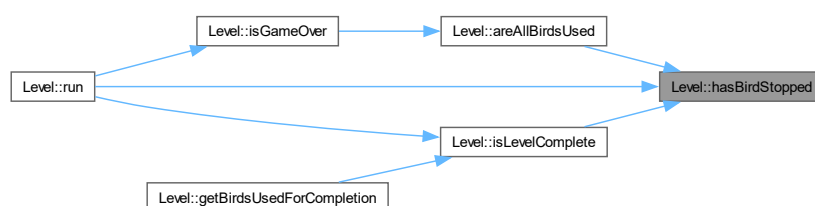
True if the bird has stopped, otherwise false.

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

<i>birdTex</i>	The texture for birds.
<i>birdTex</i>	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

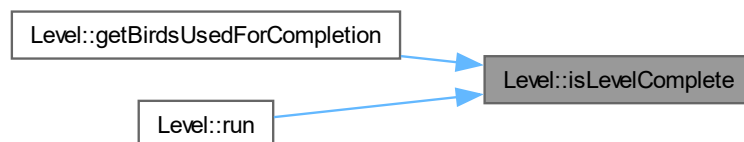
True if the level is complete, otherwise false.

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

<i>levelFile</i>	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

<i>birdTex</i>	The texture for birds.
<i>birdTex</i>	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/casptertillander/cpp-course-autumn-2023/Project/src/level.hpp`
- `//wsl.localhost/Ubuntu/home/casptertillander/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

<i>win</i>	Reference to the game's window.
<i>number</i>	The background number for the level.
<i>backTex</i>	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

<i>chosenObject</i>	The object added to the game.
<i>mousePos</i>	The position of the added object.
<i>filePath</i>	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

<i>world</i>	Pointer to the Box2D world.
<i>texture</i>	Texture for the pig.
<i>position</i>	Initial position of the pig.
<i>world</i>	Pointer to the Box2D world.
<i>texture</i>	The texture for the pig.
<i>position</i>	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

<i>window</i>	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.

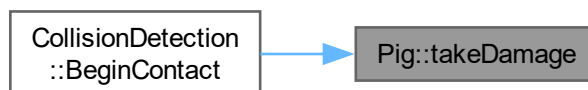
## Parameters

<i>damage</i>	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:

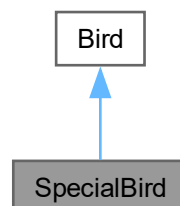
- `//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/player.hpp`
- `//wsl.localhost/Ubuntu/home/casptertillander/cpp-course-autumn-2023/Project/src/player.cpp`

## 7.11 SpecialBird Class Reference

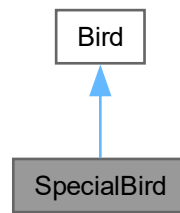
Represents a special type of bird with specific shooting behavior.

```
#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

<i>world</i>	Pointer to the Box2D world.
<i>texture</i>	The texture for the special bird.
<i>position</i>	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

<i>event</i>	The SFML event to handle.
<i>window</i>	The SFML render window.

Overrides the base class function to handle specific input for the [SpecialBird](#).

#### Parameters

<i>event</i>	The SFML event to handle.
<i>window</i>	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

<i>world</i>	A pointer to the Box2D world.
<i>texture</i>	The texture to be used for the wall sprite.
<i>position</i>	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**

<i>window</i>	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

```
00001 #ifndef BIRD_HPP
00002 #define BIRD_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <box2d/box2d.h>
00006 #include <vector>
00007
00026 class Bird {
00027 private:
00028     // Textures used in class
00029     sf::CircleShape birdShape;
00030
00031     // State variables
00032     bool isLaunched;
00033     bool isDragging;
00034     sf::Vector2f initialClickPosition;
00035
00036     // Constants
00037     const float FORCE_MULTIPLIER = 200.0f;
00038     int currentBirdIndex = 0;
00039     int totalBirds = 3;
00040
00041 public:
00042     b2Body* body;
00043
00050     Bird(b2World* world, const sf::Texture& texture, const b2Vec2& position);
00051
00055     void update();
00056
00061     void render(sf::RenderWindow& window);
00062
00068     virtual void handleInput(const sf::Event& event, const sf::RenderWindow& window);
00069
00074     void launch(const b2Vec2& force);
00075
00080     b2Vec2 getVelocity() const;
00081
00086     bool isBirdLaunched() const;
00087
00092     b2Body* getBody() const;
00093
00094     std::vector<sf::CircleShape> calculateTrajectory(const sf::Vector2f& launchVector, int numDots);
00095     std::vector<sf::CircleShape> trajectoryDots;
00096 };
00097
00098 #endif // BIRD_HPP
```

### 8.2 box.hpp

```
00001 #ifndef BOX_HPP
00002 #define BOX_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <box2d/box2d.h>
```

```

00006
00014 class Box {
00015 private:
00016     // Rendering and physics variables
00017     sf::RectangleShape boxShape;
00018     b2Body* body;
00019
00020 public:
00021     Box(b2World* world, const sf::Texture& texture, const b2Vec2& position);
00022
00023     void update();
00024
00025     void render(sf::RenderWindow& window);
00026
00027     b2Body* getBody() const;
00028     b2Vec2 getPosition() const;
00029 };
00030
00031 #endif // BOX_HPP

```

### 8.3 collisiondetection.hpp

```

00001 #ifndef COLLISIONDETECTION_HPP
00002 #define COLLISIONDETECTION_HPP
00003
00004 #include <box2d/box2d.h>
00005 #include "pig.hpp"
00006 #include "bird.hpp"
00007
00008 class CollisionDetection : public b2ContactListener {
00009 public:
00010     void BeginContact(b2Contact* contact) override;
00011
00012     bool isBirdFixture(b2Fixture* fixture);
00013 };
00014
00015 #endif // COLLISIONDETECTION_HPP

```

### 8.4 game.hpp

```

00001 #ifndef GAME_HPP
00002 #define GAME_HPP
00003
00004 #include "GUI.hpp"
00005
00006 class Game {
00007 public:
00008     Game();
00009
00010     void run();
00011
00012 private:
00013     GUI gui;
00014 };
00015
00016 #endif // GAME_HPP

```

### 8.5 GUI.hpp

```

00001 #ifndef GUI_HPP
00002 #define GUI_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include "level.hpp"
00007 #include "bird.hpp"
00008 #include "leveleditor.hpp"
00009 #include "player.hpp"
00010
00011 class GUI {
00012 public:
00013     GUI(); // Constructor

```

```

00023     void run(); // Runs the game's GUI
00024
00025     // Public members
00026     bool isSpecialBird;
00027     int selectedBackground;
00028
00029 private:
00030     // Private members
00031     // Textures
00032     sf::Texture mainScreenTexture;
00033     sf::Texture backgroundTexture;
00034     sf::Texture soundTexture;
00035     sf::Texture levelBackgroundTexture;
00036     sf::Texture settingsBackgroundTexture;
00037     sf::Texture editorBackgroundTexture;
00038     sf::Texture gameOverBackgroundTexture;
00039     sf::Texture levelCompleteBackgroundTexture;
00040     sf::Texture chooseABirdBackgroundTexture;
00041     sf::Texture birdTexture;
00042     sf::Texture specialBirdTexture;
00043     sf::Texture levelEditor1;
00044     sf::Texture levelEditor2;
00045     sf::Texture levelEditor3;
00046     sf::Texture starTexture;
00047
00048     // Fonts
00049     sf::Font font;
00050
00051     // RenderWindow
00052     sf::RenderWindow window;
00053
00054     // Music
00055     sf::Music music;
00056
00057     // Texts
00058     sf::Text titleText;
00059     sf::Text playText;
00060     sf::Text settingsText;
00061     sf::Text returnToHomeText;
00062     sf::Text levelsText;
00063     sf::Text level1Text;
00064     sf::Text level2Text;
00065     sf::Text level3Text;
00066     sf::Text tryAgainText;
00067     sf::Text returnToLevelsText;
00068     sf::Text createLevelText;
00069     sf::Text gravityText;
00070     sf::Text playerMessage;
00071     sf::Text chooseBirdText;
00072     sf::Text levelEditorText;
00073     sf::Text playerNameLabel;
00074     sf::Text submitButtonText;
00075     sf::Text inputText;
00076
00077     // Shapes and Buttons
00078     sf::CircleShape buttonShape;
00079     sf::CircleShape highlightCircle;
00080     sf::CircleShape circleButton;
00081     sf::RectangleShape highlightRectangle;
00082     sf::RectangleShape redLine;
00083     sf::RectangleShape redLine2;
00084     sf::RectangleShape playerNameInputBox;
00085     sf::RectangleShape submitButton;
00086
00087     // Sprites
00088     sf::Sprite backgroundSprite;
00089     sf::Sprite soundButton;
00090     sf::Sprite normalBirdButton;
00091     sf::Sprite specialBirdButton;
00092     sf::Sprite starSprite;
00093     sf::Sprite levelStarSprite;
00094     sf::Sprite level1Button;
00095     sf::Sprite level2Button;
00096     sf::Sprite level3Button;
00097
00098     // Game State Variables
00099     Level* currentLevel;
00100     enum Screen { Home, BirdSelection, Levels, PlayingLevel, GameOver, LevelCompleted, Settings,
LevelEditorSelection, PlayingLevelEditor };
00101     Screen currentScreen;
00102     int levelNumberEditor;
00103     LevelEditor* currentLevelEditor;
00104     std::string pathToCreatedFile;
00105     Player* currentPlayer;
00106     std::string playerNameInput;
00107
00108     // Additional int and bool members

```

```

00109     int levelNumber;
00110     sf::Vector2u textureSize;
00111     sf::Vector2u windowSize;
00112     bool soundOn;
00113     bool noGravity;
00114     bool isLevelEditorLevel;
00115
00116     // Private member functions
00117     void initialize();
00118     void processEvents();
00119     void update();
00120     void render();
00121     void startGame();
00122     void drawHomeScreen();
00123     void drawLevelsScreen();
00124     void launchLevel(int levelNumber);
00125     void drawGameOverScreen();
00126     void drawLevelCompletedScreen();
00127     void updateButtonHoverEffect(sf::Text& buttonText, sf::Vector2f mousePos);
00128     void drawSettingsScreen();
00129     void setupButton(sf::Text& buttonText, const std::string& text);
00130     void updateBackground();
00131     void drawBirdSelectionScreen();
00132     void drawLevelEditorSelectionScreen();
00133     void launchLevelEditor(int levelNumberEditor);
00134     void launchLevelEditorLevel(int levelNumberEditor, std::string filePath);
00135 };
00136
00137 #endif // GUI_HPP

```

## 8.6 level.hpp

```

00001 #ifndef LEVEL_HPP
00002 #define LEVEL_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <box2d/box2d.h>
00006 #include <vector>
00007 #include "bird.hpp"
00008 #include "specialbird.hpp"
00009 #include "pig.hpp"
00010 #include "box.hpp"
00011 #include "wall.hpp"
00012 #include "readfile.hpp"
00013 #include "collisiondetection.hpp"
00014
00024 class Level {
00025 private:
00026     // Level properties
00027     int levelNumber;
00028     int currentBirdIndex = 0;
00029     int totalBirds = 3;
00030     sf::RenderWindow& window;
00031     sf::Texture backgroundTexture;
00032     sf::Sprite backgroundSprite;
00033     b2World* world;
00034     bool noGravity;
00035
00036     // Level objects
00037     std::vector<Pig*> pigs;
00038     std::vector<Box*> boxes;
00039     std::vector<Wall*> walls;
00040     std::vector<Bird*> birds;
00041
00042     //Textures used in class
00043     sf::Texture pigTexture;
00044     sf::Texture boxTexture;
00045     sf::Texture wallTexture;
00046     sf::Texture birdTexture;
00047     sf::Font font;
00048
00049     // UI elements
00050     sf::Text birdsRemainingText;
00051     sf::Text pigsRemainingText;
00052
00053 public:
00063     Level(sf::RenderWindow& win, int number, const sf::Texture& backTex, const std::string& levelFile,
00064           bool isSpecialBird, bool noGravity);
00065
00066     bool isSpecialBird;
00070     ~Level();
00071

```

```

00075     void run();
00076
00080     void setupWorld();
00081
00085     void createFloor();
00086
00090     void createBoundaries();
00091
00100     void createBoundary(float x, float y, float width, float height);
00101
00107     void loadObjects(const std::string& levelFile);
00108
00114     void initializeBirds(const sf::Texture& birdTex, bool isSpecialBird);
00115
00121     void nextBird(const sf::Texture& birdTex, bool isSpecialBird);
00122
00128     bool hasBirdStopped() const;
00129
00135     bool isLevelComplete() const;
00136
00142     bool isGameOver() const;
00143
00149     bool areAllPigsDestroyed() const;
00150
00156     bool areAllBirdsUsed() const;
00157
00158     int getBirdsUsedForCompletion();
00159
00160     Bird* getCurrentBird();
00161
00162 };
00163
00164 #endif // LEVEL_HPP
00165
00166
00167
00168
00169

```

## 8.7 leveleditor.hpp

```

00001 #ifndef LEVEL_EDITOR_HPP
00002 #define LEVEL_EDITOR_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <box2d/box2d.h>
00006
00012 class LevelEditor {
00013 private:
00014     // Member variables
00015     int levelNumber;
00016     int chosenObject = 1;
00017
00018     sf::Text playCreatedLevelText;
00019     sf::Font font;
00020     sf::CircleShape ButtonShape;
00021
00022     sf::RenderWindow& window;
00023     sf::Texture backgroundTexture;
00024     sf::Sprite backgroundSprite;
00025
00026     sf::Texture pigTexture;
00027     sf::Texture wallTexture;
00028     sf::Texture boxTexture;
00029
00030     sf::Sprite pigSprite;
00031     sf::Sprite wallSprite;
00032     sf::Sprite boxSprite;
00033
00034     sf::CircleShape pigShape;
00035     sf::RectangleShape boxShape;
00036     sf::RectangleShape wallShape;
00037
00038     std::vector<sf::Vector2f> pigPositions;
00039     std::vector<sf::Vector2f> boxPositions;
00040     std::vector<sf::Vector2f> wallPositions;
00041
00042     sf::RectangleShape highlightRectangle;
00043
00044     std::string filePath = "../Createdlevels/createdlevel.csv";
00045
00046 public:
00054     LevelEditor(sf::RenderWindow& win, int number, const sf::Texture& backTex);

```

```

00055
00059     std::string run();
00060
00064     void setUpLevel();
00065
00072     void addObject(int chosenObject, sf::Vector2f mousePos, std::string& filePath);
00073
00077     void drawObject();
00078
00079 };
00080
00081
00082 #endif // LEVEL_EDITOR_HPP

```

## 8.8 pig.hpp

```

00001 #ifndef PIG_HPP
00002 #define PIG_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <box2d/box2d.h>
00006
00012 class Pig {
00013 private:
00014     // Rendering and physics variables
00015     sf::CircleShape pigShape;
00016     b2Body* body;
00017
00018     // State variables
00019     float health;
00020     bool isAlive;
00021     bool markedForDeletion;
00022
00023 public:
00031     Pig(b2World* world, const sf::Texture& texture, const b2Vec2& position);
00032
00036     void update();
00037
00043     void render(sf::RenderWindow& window);
00044
00050     void takeDamage(float damage);
00051
00057     bool alive() const;
00058
00062     void destroyBody();
00063
00067     void markForDeletion();
00068
00074     bool isMarkedForDeletion() const;
00075
00081     float getHealth() const;
00082
00088     b2Body* getBody() const;
00089 };
00090
00091 #endif // PIG_HPP

```

## 8.9 player.hpp

```

00001 #ifndef PLAYER_HPP
00002 #define PLAYER_HPP
00003
00004 #include <string>
00005 #include <map>
00006
00012 class Player {
00013 public:
00014     // Player name
00015     std::string name;
00016
00021     std::map<int, int> levelScores;
00022
00028     Player(std::string name);
00029
00035     std::string loadFromFile();
00036
00040     void saveToFile();
00041
00048     int getScoreForLevel(int levelNumber);

```



```

00049
00056     void updateScore(int levelNumber, int stars);
00057 };
00058
00059 #endif // PLAYER_HPP

```

## 8.10 readfile.hpp

```

00001 #ifndef READFILE_HPP
00002 #define READFILE_HPP
00003
00004 #include <string>
00005 #include <vector>
00006
00012 struct ObjectData {
00014     std::string type;
00015
00016     // Coordinates
00017     float x, y;
00018 };
00019
00026 std::vector<ObjectData> readLevelData(const std::string& filename);
00027
00028 #endif // READFILE_HPP
00029

```

## 8.11 specialbird.hpp

```

00001 #ifndef SPECIALBIRD_HPP
00002 #define SPECIALBIRD_HPP
00003
00004 #include "bird.hpp"
00005
00017 class SpecialBird : public Bird {
00018 public:
00019     SpecialBird(b2World* world, const sf::Texture& texture, const b2Vec2& position);
00020
00027     void handleInput(const sf::Event& event, const sf::RenderWindow& window) override;
00028
00034     bool getShot();
00035
00036 private:
00042     void shootTowardsClick(const sf::Vector2f& targetPosition);
00043
00044     bool isShot;
00045 };
00046
00047 #endif // SPECIALBIRD_HPP

```

## 8.12 wall.hpp

```

00001 #ifndef WALL_HPP
00002 #define WALL_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <b2d/box2d.h>
00006
00013 class Wall {
00014 private:
00015     sf::RectangleShape wallShape;
00016     b2Body* body;
00017
00018 public:
00026     Wall(b2World* world, const sf::Texture& texture, const b2Vec2& position);
00027
00031     void update();
00032
00038     void render(sf::RenderWindow& window);
00039
00045     b2Body* getBody() const;
00046
00047
00053     b2Vec2 getPosition() const;
00054 };
00055
00056 #endif // WALL_HPP
00057

```



# Index

`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/GUI.hpp`, 46  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/bird.hpp`, 45  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/box.hpp`, 45  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/collisiondetection.hpp`, 46  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/game.hpp`, 46  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/level.hpp`, 48  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/leveleditor.hpp`, 49  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/pig.hpp`, 50  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/player.hpp`, 50  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/readfile.hpp`, 51  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/specialbird.hpp`, 51  
`//wsl.localhost/Ubuntu/home/casptillander/cpp-course-autumn-2023/Project/src/wall.hpp`, 51

`addObject`  
    `LevelEditor`, 34

`alive`  
    `Pig`, 37

`areAllBirdsUsed`  
    `Level`, 28

`areAllPigsDestroyed`  
    `Level`, 28

`BeginContact`  
    `CollisionDetection`, 23

`Bird`, 15  
    `Bird`, 16  
    `calculateTrajectory`, 17  
    `getBody`, 17  
    `getVelocity`, 17  
    `handleInput`, 18  
    `isBirdLaunched`, 19  
    `launch`, 19  
    `render`, 20  
    `update`, 20

`Box`, 20  
    `Box`, 21  
    `getBody`, 21  
    `getPosition`, 21  
    `render`, 22  
    `update`, 22

`calculateTrajectory`  
    `Bird`, 17

`CollisionDetection`, 22  
    `BeginContact`, 23  
    `isBirdFixture`, 24

`createBoundary`  
    `Level`, 29

`destroyBody`  
    `Pig`, 37

`Game`, 25  
`getBirdsUsedForCompletion`  
    `Level`, 29

`getBody`  
    `Bird`, 17  
    `Box`, 21  
    `Pig`, 37  
    `Wall`, 43

`getHealth`  
    `Pig`, 37

`getPosition`  
    `Box`, 21  
    `Wall`, 43

`getScoreForLevel`  
    `Level Properties`, 11

`getShot`  
    `SpecialBird`, 41

`getVelocity`  
    `Bird`, 17

`GUI`, 25

`handleInput`  
    `Bird`, 18  
    `SpecialBird`, 41

`hasBirdStopped`  
    `Level`, 30

`initializeBirds`  
    `Level`, 30

`isBirdFixture`

- CollisionDetection, 24
- isBirdLaunched
  - Bird, 19
- isGameOver
  - Level, 31
- isLevelComplete
  - Level, 32
- isMarkedForDeletion
  - Pig, 37
- launch
  - Bird, 19
- Level, 26
  - areAllBirdsUsed, 28
  - areAllPigsDestroyed, 28
  - createBoundary, 29
  - getBirdsUsedForCompletion, 29
  - hasBirdStopped, 30
  - initializeBirds, 30
  - isGameOver, 31
  - isLevelComplete, 32
  - Level, 27
  - loadObjects, 32
  - nextBird, 33
- Level Properties, 11
  - getScoreForLevel, 11
  - loadFromFile, 12
  - Player, 12
  - updateScore, 12
- LevelEditor, 34
  - addObject, 34
  - LevelEditor, 34
- loadFromFile
  - Level Properties, 12
- loadObjects
  - Level, 32
- nextBird
  - Level, 33
- ObjectData, 35
- Pig, 36
  - alive, 37
  - destroyBody, 37
  - getBody, 37
  - getHealth, 37
  - isMarkedForDeletion, 37
  - Pig, 36
  - render, 37
  - takeDamage, 38
  - update, 38
- Player, 39
  - Level Properties, 12
- render
  - Bird, 20
  - Box, 22
  - Pig, 37
  - Wall, 43
- Source content, 1
- SpecialBird, 13, 39
  - getShot, 41
  - handleInput, 41
  - SpecialBird, 41
- takeDamage
  - Pig, 38
- update
  - Bird, 20
  - Box, 22
  - Pig, 38
- updateScore
  - Level Properties, 12
- Wall, 42
  - getBody, 43
  - getPosition, 43
  - render, 43
  - Wall, 43