Angry_Birds_Alike

Generated by Doxygen 1.10.0

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Entity Class Reference

Represents an entity in the game, such as a projectile or target.

`#include <Entity.h>`

Public Member Functions

- Entity (float radius, sf::Color color, double m, bool mobile, bool gravity)
- void applyForce (float forceMagnitude, float angleDegrees)
- void update (float deltaTime, Stepper &stepper)
- void resetPosition (float x, float y)
- void transferMomentum (Entity &other)

Static Public Member Functions

- static float degreesToRadians (float degrees)

  Converts an angle from degrees to radians.

Public Attributes

- bool collisionFlag = false
- sf::CircleShape **shape**

  The shape representing the entity in the simulation.
- State **state**

  The current state of the entity (x, y, vx, vy).
- bool **isMobile**

  Flag indicating if the entity can move.
- bool **affectedByGravity**

  Flag indicating if the entity is affected by gravity.
- double **mass**

  The mass of the entity.

3.1.1 Detailed Description

Represents an entity in the game, such as a projectile or target.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Entity()

```
Entity::Entity (
            float radius,
            sf::Color color,
            double m,
            bool mobile,
            bool gravity )
```

Constructs an Entity with specified properties.

Parameters

radius	The radius of the entity.
color	The color of the entity.
m	The mass of the entity.
mobile	Indicates if the entity is mobile.
gravity	Indicates if the entity is affected by gravity.

3.1.3 Member Function Documentation

3.1.3.1 applyForce()

```
void Entity::applyForce (
            float forceMagnitude,
            float angleDegrees )
```

Applies a force to the entity, modifying its velocity.

Parameters

forceMagnitude	The magnitude of the force.
angleDegrees	The direction of the force in degrees.

Applies a force to the entity, modifying its velocity based on the force magnitude and direction. The force is directly converted to initial velocity components using trigonometric calculations.

3.1.3.2 degreesToRadians()

```
static float Entity::degreesToRadians (
            float degrees )  [inline], [static]
```

Converts an angle from degrees to radians.

Parameters

degrees	The angle in degrees to convert.

Returns

The angle converted to radians.

3.1.3.3 resetPosition()

```
void Entity::resetPosition (
            float x,
            float y )
```

Resets the entity's position to specified coordinates.

Parameters

x	The x-coordinate of the new position.
y	The y-coordinate of the new position.

Resets the entity's position to specified coordinates, making it immobile and unaffected by gravity.

3.1.3.4 transferMomentum()

```
void Entity::transferMomentum (
            Entity & other )
```

Transfers momentum between this entity and another during a collision, following the laws of conservation of momentum. Assumes a perfectly elastic collision for simplicity.

3.1.3.5 update()

```
void Entity::update (
            float deltaTime,
            Stepper & stepper )
```

Updates the entity's position based on its velocity.

Parameters

deltaTime	The time elapsed since the last update.
stepper	The numerical stepper used for integration.

Updates the entity's position based on its velocity. Uses the numerical integration stepper to compute the new state of the entity over time.

3.1.4 Member Data Documentation

3.1.4.1 collisionFlag

```
bool Entity::collisionFlag = false
```

Transfers momentum between this entity and another during a collision.

Parameters

other	The other entity involved in the collision.

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

- include/Entity.h
- src/Entity.cpp

3.2 Game Class Reference

Manages the game logic, user input, and rendering.

```
#include <Game.h>
```

Public Member Functions

- **Game** ()
  Constructs a new Game instance.
- Game (const int maxBirdResets)
  Constructs a new Game instance with specified settings.
- void run ()

3.2.1 Detailed Description

Manages the game logic, user input, and rendering.

This class is responsible for initializing the game, handling user input, updating game state, and rendering graphics to the screen.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Game()

```
Game::Game (
            const int maxBirdResets )
```

Constructs a new Game instance with specified settings.

Parameters

maxBirdResets	The maximum number of times the bird can be reset in the game.

3.2.3 Member Function Documentation

3.2.3.1 run()

```
void Game::run ( )
```

Runs the main game loop.

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

- include/Game.h
- src/Game.cpp

Chapter 4

File Documentation

4.1 include/Entity.h File Reference

Represents a physical entity in the simulation.

```
#include <SFML/Graphics.hpp>
#include <array>
#include <boost/numeric/odeint/stepper/runge_kutta4.hpp>
#include <boost/numeric/odeint.hpp>
#include "/Users/jubair/CLionProjects/untitled/include/Physics.h"
#include <vector>
```

Classes

- class Entity

  Represents an entity in the game, such as a projectile or target.

Typedefs

- using State = std::array<double, 4>

  Defines the state of an entity within the game.
- using Stepper = boost::numeric::odeint::runge_kutta4<State>

  Represents the numerical stepper used for physics integration within the game.

4.1.1 Detailed Description

Represents a physical entity in the simulation.

This class encapsulates all the properties of a physical entity, including its mass, position, velocity, and provides methods to manipulate these properties.

4.1.2 Typedef Documentation

4.1.2.1 State

```
using State = std::array<double, 4>
```

Defines the state of an entity within the game.

This type is used to represent the position and velocity of entities.

4.1.2.2 Stepper

```
using Stepper = boost::numeric::odeint::runge_kutta4<State>
```

Represents the numerical stepper used for physics integration within the game.

This stepper is utilized to advance the state of entities based on the physics simulation.

4.2 Entity.h

Go to the documentation of this file.

```
00001
00009 //
00010 // Created by Jubair on 2024-03-25.
00011 //
00012
00013 #ifndef UNTITLED_ENTITY_H
00014 #define UNTITLED_ENTITY_H
00015
00016 #ifndef ENTITY_HPP
00017 #define ENTITY_HPP
00018
00019 #include <SFML/Graphics.hpp>
00020 #include <array>
00021 #include <boost/numeric/odeint/stepper/runge_kutta4.hpp>
00022 #include <boost/numeric/odeint.hpp>
00023 #include "/Users/jubair/CLionProjects/untitled/include/Physics.h"
00024 #include <vector>
00025
00031 using State = std::array<double, 4>; // Replace with the actual line 29 content.
00032
00038 using Stepper = boost::numeric::odeint::runge_kutta4<State>; // Replace with the actual line 35
      content.
00039
00044 class Entity {
00045 public:
00056     Entity(float radius, sf::Color color, double m, bool mobile, bool gravity);
00063     void applyForce(float forceMagnitude, float angleDegrees);
00070     void update(float deltaTime, Stepper& stepper);
00077     void resetPosition(float x, float y);
00083     bool collisionFlag = false; // New flag to indicate a recent collision
00084     void transferMomentum(Entity &other);
00085
00086     // Member variables are typically documented where they are declared.
00087     sf::CircleShape shape;
00088     State state;
00089     bool isMobile;
00090     bool affectedByGravity;
00091     double mass;
00097     static float degreesToRadians(float degrees) {
00098         return degrees * PI / 180.0f;
00099         }
00100
00101 };
00102
00103 #endif // ENTITY_HPP
00104
00105 #endif //UNTITLED_ENTITY_H
```

4.3 include/Game.h File Reference

Main game interface for handling game logic and rendering.

```
#include <SFML/Graphics.hpp>
#include "/Users/jubair/CLionProjects/untitled/include/Entity.h"
#include <boost/numeric/odeint.hpp>
#include <vector>
#include <thread>
```

Classes

- class Game

  Manages the game logic, user input, and rendering.

Typedefs

- using State = std::array<double, 4>

  Defines the state of an entity within the game.
- using Stepper = boost::numeric::odeint::runge_kutta4<State>

  Represents the numerical stepper used for physics integration within the game.

4.3.1 Detailed Description

Main game interface for handling game logic and rendering.

This file defines the Game class, which is responsible for managing the game loop, handling user input, simulating physics, and rendering entities and other game elements.

4.3.2 Typedef Documentation

4.3.2.1 State

```
using State = std::array<double, 4>
```

Defines the state of an entity within the game.

This type is used to represent the position and velocity of entities.

4.3.2.2 Stepper

```
using Stepper = boost::numeric::odeint::runge_kutta4<State>
```

Represents the numerical stepper used for physics integration within the game.

This stepper is utilized to advance the state of entities based on the physics simulation.

4.4 Game.h

Go to the documentation of this file.

```
00001
00009 //
00010 // Created by Jubair on 2024-03-25.
00011 //
00012
00013 #ifndef UNTITLED_GAME_H
00014 #define UNTITLED_GAME_H
00015
00016 #ifndef GAME_HPP
00017 #define GAME_HPP
00018
00019 #include <SFML/Graphics.hpp>
00020 #include "/Users/jubair/CLionProjects/untitled/include/Entity.h"
00021 #include <boost/numeric/odeint.hpp>
00022 #include <vector>
00023 #include <thread>
00024
00030 using State = std::array<double, 4>; // Replace with the actual line 29 content.
00031
00037 using Stepper = boost::numeric::odeint::runge_kutta4<State>; // Replace with the actual line 35
      content.
00038
00046 class Game {
00047 public:
00048
00052     Game();
00057     Game(const int maxBirdResets);
00061     void run();
00062 private:
00068     void handleKeyPress(sf::Keyboard::Key key);
00069
00073     void update();
00081     static bool checkCollision(const Entity& a, const Entity& b);
00088     void simulateTrajectory(float degrees, float speed);
00095     void updateLaunchArrow(float angleDegrees, float force);
00099     void render();
00103     void resetGame();
00107     void resetBirdPosition();
00114     static bool isOutOfWindow(const Entity& entity);
00122     void printVelocities(const std::string& phase, const Entity& a, const Entity& b);
00123
00124     static float launchAngleDegrees;
00125     static float force;
00126
00127     sf::RenderWindow window;
00128     Entity projectile;
00129     std::vector<Entity> targets;
00130     Stepper stepper;
00131     sf::VertexArray trajectoryLine;
00132     sf::RectangleShape launchArrow;
00133     sf::Clock clock;
00134     int birdResetCounter = 0;
00135     const int maxBirdResets;
00136
00137 };
00138
00139 #endif // GAME_HPP
00140
00141
00142 #endif //UNTITLED_GAME_H
```

4.5 include/Physics.h File Reference

Physics system for projectiles, including constants and basic physics operations.

```
#include "/Users/jubair/CLionProjects/untitled/include/State.h"
```

Functions

- void projectileSystem (const State &state, State &dstate_dt, double time)

  Updates the derivative of the state for projectile motion.

Variables

- const float **PI** = 3.14159265f

  The value of pi.
- const float **GRAVITY** = -981.0f

  Acceleration due to gravity (cm/s^ 2).

4.5.1 Detailed Description

Physics system for projectiles, including constants and basic physics operations.

Defines the physics system responsible for updating the state of entities in the simulation. It includes fundamental physics constants and the function prototype for the projectile system.

4.5.2 Function Documentation

4.5.2.1 projectileSystem()

```
void projectileSystem (
            const State & state,
            State & dstate_dt,
            double time )
```

Updates the derivative of the state for projectile motion.

Parameters

state	The current state of the entity.
dstate↩ _dt	The derivative of the state to be updated.
time	The current time step for the simulation.

4.6 Physics.h

Go to the documentation of this file.

```
00001
00009 //
00010 // Created by Jubair on 2024-03-25.
00011 //
00012
00013 #ifndef UNTITLED_PHYSICS_H
00014 #define UNTITLED_PHYSICS_H
00015
00016 #ifndef PHYSICS_HPP
00017 #define PHYSICS_HPP
00018 #include "/Users/jubair/CLionProjects/untitled/include/State.h"
00019
00020 const float PI = 3.14159265f;
00021 const float GRAVITY = -981.0f;
00022
00023 class Entity;
00024
00031 // Physics system for the projectile
00032 void projectileSystem(const State &state, State &dstate_dt, double time);
00033 #endif // PHYSICS_HPP
00034
00035 #endif //UNTITLED_PHYSICS_H
```

4.7 include/State.h File Reference

Defines the State type used throughout the simulation.

```
#include <array>
#include <thread>
```

Typedefs

- using State = std::array<double, 4>

4.7.1 Detailed Description

Defines the State type used throughout the simulation.

This header file introduces a State type, representing the fundamental state (position and velocity) of entities within the simulation. It is utilized across various components of the project for tracking and updating entity states.

4.7.2 Typedef Documentation

4.7.2.1 State

```
using State = std::array<double, 4>
```

Represents the state of an entity in terms of its position and velocity. The state is stored as an array of doubles, where indices correspond to: 0: x position, 1: y position, 2: x velocity, 3: y velocity.

4.8 State.h

Go to the documentation of this file.

```
00001
00011 //
00012 // Created by Jubair on 2024-03-25.
00013 //
00014
00015 #ifndef UNTITLED_STATE_H
00016 #define UNTITLED_STATE_H
00017 #include <array>
00018 #include <thread>
00024 using State = std::array<double, 4>; // x, y, vx, vy
00025 #endif //UNTITLED_STATE_H
```

4.9 src/main.cpp File Reference

Entry point for the simulation game.

```
#include "/Users/jubair/CLionProjects/untitled/include/Game.h"
```

Functions

- int main ()

4.9.1 Detailed Description

Entry point for the simulation game.

Initializes and runs the game.

4.9.2 Function Documentation

4.9.2.1 main()

```
int main ( )
```

Main function.

Returns

int Exit status code. Returns 0 on successful execution.