Stickman Fighter

4

Generated by Doxygen 1.8.16

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 stickman Namespace Reference

5.1.1 *

Classes

- class AssetManager

  Class for asset manager.This class loads a texture and creates a map between textures and strings so that we don't have to load the same texture and sprite again and again.
- struct emptyStr

  An empty struct to derive from.
- class Game

  Contains all basic entities required in game like window, players object, TcpListener, Send and receive sockets, Box2D world, walls,ground, sprites of all bodies to be displayed in window and functions to check collision, decrease health points, sending and receving packets from client to server and vice versa, worker threads which checks for collision and gem thread used to generate gem.
- class Game2

  Class for game which initializes the different properties related to the game like resolution and so on.
- struct GameData

  This contains the objects required by the game as the whole like the state machine which switches states and different managers to make loading different things easier.
- class GameOver

  Class for game over state.
- class HelpState

  Class for help state.
- class InputManager

  Class for input manager.
- struct mainGame

  Class for main game.
- class MainMenuState

  Class for main menu state.
- class myListener

  Listens to collision between any two objects in Box2D world.
- class NameState

Class for name state which takes the name of player and gives the option of chosing whether to host a server/ join a server.
- class Player

    Contains all the information about the player.
- struct playerdata

    Struct for testing player data.
- class SplashState

    Class for splash state.
- class State

    Class for state which has functions which can be overloaded so that a particular state of the game can run using these functions.
- class StateMachine

    Class which is responsible for running a state when it gets loaded.
- struct testDecreaseHp

    This will be passed to the test as we want an interface to the previous struct.

5.1.2 *

Typedefs

- typedef std::shared_ptr< GameData > GameDataRef

    Creating container for raw pointers for the struct game data.
- typedef std::unique_ptr< State > StateRef

    Creates a unique pointer for StateRef so that it gets automatically destroyed.

5.1.3 *

Functions

- TEST_F (mainGame, initializeData)

    Runs the test initializeData.
- TEST_F (mainGame, generateGem)

    test for generate gem function
- TEST_F (mainGame, checkDistance)

    Test for the distance function.
- TEST_P (testDecreaseHp, decreaseHp)

    Performs a test with multiple inputs to check different test cases.
- INSTANTIATE_TEST_CASE_P (Default, testDecreaseHp, testing::Values(playerdata{100, 100, 1, 5}, playerdata{100, 100, 5, 1}, playerdata{100, 100, 4, 6}, playerdata{100, 100, 6, 4}, playerdata{100, 100, 6, 3}, playerdata{100, 100, 3, 6}, playerdata{100, 100, 4, 5}, playerdata{100, 100, 5, 4}, playerdata{100, 100, 3, 5}, playerdata{100, 100, 5, 3}, playerdata{100, 100, 2, 8}, playerdata{100, 100, 8, 2}, playerdata{100, 100, 7, 2}, playerdata{100, 100, 2, 7}, playerdata{100, 100, 1, 7}, playerdata{100, 100, 7, 1}, playerdata{80, 100, 4, 6}, playerdata{100, 80, 6, 4}, playerdata{100, 60, 4, 6}, playerdata{60, 100, 6, 4}, playerdata{100, 40, 4, 6}, playerdata{40, 100, 6, 4}, playerdata{100, 20, 4, 6}, playerdata{20, 100, 6, 4}, playerdata{100, 10, 4, 6}, playerdata{10, 100, 6, 4}, playerdata{80, 80, 4, 6}, playerdata{80, 80, 6, 4}, playerdata{80, 60, 4, 6}, playerdata{60, 80, 6, 4}, playerdata{80, 40, 4, 6}, playerdata{40, 80, 6, 4}, playerdata{80, 20, 4, 6}, playerdata{20, 80, 6, 4}, playerdata{80, 10, 4, 6}, playerdata{10, 80, 6, 4}, playerdata{60, 80, 4, 6}, playerdata{80, 60, 6, 4}, playerdata{60, 60, 4, 6}, playerdata{60, 60, 6, 4}, playerdata{60, 40, 4, 6}, playerdata{40, 60, 6, 4}, playerdata{60, 20, 4, 6}, playerdata{20, 60, 6, 4}, playerdata{60, 10, 4, 6}, playerdata{10, 60, 6, 4}, playerdata{40, 80, 4, 6}, playerdata{80, 40, 6, 4}, playerdata{40, 60, 4, 6}, playerdata{60, 40, 6, 4}))

    Passes the test cases for the test.

5.1.4 Typedef Documentation

5.1.4.1 GameDataRef

```
typedef std::shared_ptr<GameData> stickman::GameDataRef
```

Creating container for raw pointers for the struct game data.

5.1.4.2 StateRef

```
typedef std::unique_ptr<State> stickman::StateRef
```

Creates a unique pointer for StateRef so that it gets automatically destroyed.

5.1.5 Function Documentation

5.1.5.1 INSTANTIATE_TEST_CASE_P()

```
stickman::INSTANTIATE_TEST_CASE_P (
             Default ,
             testDecreaseHp ,
             testing::Values(playerdata{100, 100, 1, 5}, playerdata{100, 100, 5, 1}, playerdata{100,
100, 4, 6}, playerdata{100, 100, 6, 4}, playerdata{100, 100, 6, 3}, playerdata{100, 100, 3,
6}, playerdata{100, 100, 4, 5}, playerdata{100, 100, 5, 4}, playerdata{100, 100, 3, 5}, playerdata{100,
100, 5, 3}, playerdata{100, 100, 2, 8}, playerdata{100, 100, 8, 2}, playerdata{100, 100, 7,
2}, playerdata{100, 100, 2, 7}, playerdata{100, 100, 1, 7}, playerdata{100, 100, 7, 1}, playerdata{80,
100, 4, 6}, playerdata{100, 80, 6, 4}, playerdata{100, 60, 4, 6}, playerdata{60, 100, 6, 4},
playerdata{100, 40, 4, 6}, playerdata{40, 100, 6, 4}, playerdata{100, 20, 4, 6}, playerdata{20,
100, 6, 4}, playerdata{100, 10, 4, 6}, playerdata{10, 100, 6, 4}, playerdata{80, 80, 4, 6},
playerdata{80, 80, 6, 4}, playerdata{80, 60, 4, 6}, playerdata{60, 80, 6, 4}, playerdata{80,
40, 4, 6}, playerdata{40, 80, 6, 4}, playerdata{80, 20, 4, 6}, playerdata{20, 80, 6, 4}, playerdata{80,
10, 4, 6}, playerdata{10, 80, 6, 4}, playerdata{60, 80, 4, 6}, playerdata{80, 60, 6, 4}, playerdata{60,
60, 4, 6}, playerdata{60, 60, 6, 4}, playerdata{60, 40, 4, 6}, playerdata{40, 60, 6, 4}, playerdata{60,
20, 4, 6}, playerdata{20, 60, 6, 4}, playerdata{60, 10, 4, 6}, playerdata{10, 60, 6, 4}, playerdata{40,
80, 4, 6}, playerdata{80, 40, 6, 4}, playerdata{40, 60, 4, 6}, playerdata{60, 40, 6, 4})  )
```

Passes the test cases for the test.

5.1.5.2 TEST_F() [1/3]

```
stickman::TEST_F (
             mainGame ,
             checkDistance  )
```

Test for the distance function.

5.1.5.3 TEST_F() [2/3]

```
stickman::TEST_F (
            mainGame ,
            generateGem  )
```

test for generate gem function

5.1.5.4 TEST_F() [3/3]

```
stickman::TEST_F (
            mainGame ,
            initializeData  )
```

Runs the test initializeData.

5.1.5.5 TEST_P()

```
stickman::TEST_P (
            testDecreaseHp ,
            decreaseHp  )
```

Performs a test with multiple inputs to check different test cases.

Chapter 6

Class Documentation

6.1 stickman::AssetManager Class Reference

Class for asset manager.This class loads a texture and creates a map between textures and strings so that we don't have to load the same texture and sprite again and again.

```
#include <AssetManager.hpp>
```

6.1.1 *

Public Member Functions

- AssetManager ()

  Constructs the object.
- ∼AssetManager ()

  Destroys the object.
- void LoadTexture (std::string name, std::string fileName)

  Loads a texture and maps it to a string.
- sf::Texture & GetTexture (std::string name)

  Gets a texture by its name as specified in the dictionary.
- void LoadFont (std::string name, std::string fileName)

  Loads a font and maps it to a string.
- sf::Font & GetFont (std::string name)

  Gets a font by its name as specified in the dictionary.

6.1.2 Detailed Description

Class for asset manager.This class loads a texture and creates a map between textures and strings so that we don't have to load the same texture and sprite again and again.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 AssetManager()

```
stickman::AssetManager::AssetManager ( )  [inline]
```

Constructs the object.

6.1.3.2 ∼AssetManager()

```
stickman::AssetManager::∼AssetManager ( )  [inline]
```

Destroys the object.

6.1.4 Member Function Documentation

6.1.4.1 GetFont()

```
sf::Font & stickman::AssetManager::GetFont (
             std::string name )
```

Gets a font by its name as specified in the dictionary.

Parameters

in	*name*	The name of the font to fetch

Returns

The font.

6.1.4.2 GetTexture()

```
sf::Texture & stickman::AssetManager::GetTexture (
             std::string name )
```

Gets a texture by its name as specified in the dictionary.

Parameters

in	*name*	The name of the texture to fetch

Returns

The texture.

6.1.4.3 LoadFont()

```
void stickman::AssetManager::LoadFont (
            std::string name,
            std::string fileName )
```

Loads a font and maps it to a string.

Parameters

in	*name*	The string with which it will be mapped.
in	*fileName*	The string which has the filepath to the font.

6.1.4.4 LoadTexture()

```
void stickman::AssetManager::LoadTexture (
            std::string name,
            std::string fileName )
```

Loads a texture and maps it to a string.

Parameters

in	*name*	The string with which it will be mapped.
in	*fileName*	The string which has the filepath to the texture.

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

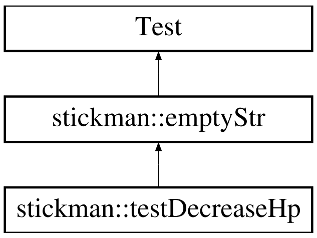
- AssetManager.hpp
- AssetManager.cpp

6.2 stickman::emptyStr Struct Reference

An empty struct to derive from.

```
#include <tests.hpp>
```

Inheritance diagram for stickman::emptyStr:



6.2.1 Detailed Description

An empty struct to derive from.

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

- tests.hpp

6.3 stickman::Game Class Reference

Contains all basic entities required in game like window, players object, TcpListener, Send and receive sockets, Box2D world, walls,ground, sprites of all bodies to be displayed in window and functions to check collision, decrease health points, sending and receving packets from client to server and vice versa, worker threads which checks for collision and gem thread used to generate gem.

```
#include <game.h>
```

6.3.1 *

Public Member Functions

- Game (GameDataRef data, std::string s, bool client, std::string myip)

  Constructor for game.
- ∼Game ()

  Destructor for game.
- b2Body ∗ createGround (b2Vec2 position, int angle)

  It is used to create ground Box2D body in the Box2D world.
- void initPlayer (Player ∗player, float X, int offset)

  Initialization of player object.
- void gameLoop ()

  Game loop which runs until game is finished.
- void updatePlayer (Player ∗player)

  It is used to update positions and angle of all body parts of each player after each time frame.
- void draw (Player ∗player)

  It is used to draw different body parts like head, body, hands and legs of player in window.
- void checkcollision ()

  It checks collision between body parts of first player and second player.
- void decrease_hp (int a, int b)

  It is used to decrease health point of player according to collision of different body parts of each player.
- void generateGem ()

It generates gem at random positions in the window after every 5 seconds and checks if any player has collected gem and accordingly increases its health points by 5.

- float distance (int x1, int y1, int x2, int y2)

  Caclulates distance between two points in the window.
- void server_send ()

  Used to send packets of information like position of all body parts of each player and position of gem from server side to client side.
- void server_receive ()

  Used to receive packets of information in server side of the key pressed by client so that server can simulate it in its world.
- void client_send ()

  Used to send packets of information like the key pressed by client to move the player in client side to server side.
- void client_receive (float ∗x, float ∗y, float ∗angle, int ∗hp, float ∗gempos)

  Used to reveive packets of information like both player's position, angle, hp and gem's position in client side from server side.
- void connect ()

  In server side, TCP listener listens to client on first and second port and on client side, it is used to connect to IP address of server and the two ports.
- void destroyBody ()

  Used to destroy Box2D bodies that is head, body, right and left hand and legs of player object.
- void serverListen (bool flag)

  TCP listener on server side listens if client has connected to any port on server's IP Address.
- int getPlayerRounds (bool player)

  It returns number of rounds won by player.
- void setPlayerRounds (bool player, int rounds)

  Sets the number of rounds won by player.

6.3.2 *

Public Attributes

- sf::RenderWindow ∗ window

  The main SFML window of type RenderWindow over which game is displayed.
- sf::IpAddress ip

  Denotes the IP Address over which server hosts.
- sf::TcpListener tcplistener

  Basically, it is a TCP listener listens to a particular port and accepts if client connects.
- sf::TcpListener tcplistener1

  It is a TCP listener whichlistens to a particular port and accepts if client connects.
- sf::TcpSocket sendSocket

  It is a TCP Socket used for sending packets from server side and receiving packets in client side.
- sf::TcpSocket listenSocket

  It is a TCP Socket used for sending packets from client side and receving packets in client side.
- b2World ∗ world

  It is Box2D world object pointer where all the bodies of player reside.
- b2Body ∗ ground

  It is a Box2D body object pointer which is denoting the ground situated in world.
- b2Body ∗ wall1

  It is a Box2D body object pointer which is denoting the upper wall situated in world.
- b2Body ∗ wall2

  It is a Box2D body object pointer which is denoting the left most wall situated in world.

- b2Body ∗ wall3

  It is a Box2D body object pointer which is denoting the right most wall situated in world.
- std::string myip

  It is a string containing IP Address of server.
- int groundUserData

  Contains the userData of ground.
- sf::Texture groundTexture

  Denotes texture of ground where images of ground is to loaded.
- sf::Texture wall1Texture

  Denotes texture of upper wall where image of ground is to loaded.
- sf::Texture wall2Texture

  Denotes texture of left most wall where image of wall is to loaded.
- sf::Texture wall3Texture

  Denotes texture of right most wall where image of wall is to loaded.
- sf::Texture roundTexture

  Denotes texture of background where number of rounds won by each player is displayed.
- sf::Texture gemTexture

  Denotes texture of gem which gets generated in the game every 5 seconds.
- sf::Sprite groundSprite

  Denotes Sprite of ground used to display ground in window.
- sf::Sprite wall1Sprite

  Denotes Sprite of upper wall used to display upper wall in window.
- sf::Sprite wall2Sprite

  Denotes Sprite of lefmost wall used to display leftmost wall in window.
- sf::Sprite wall3Sprite

  Denotes Sprite of rightmost wall used to display rightmost wall in window.
- sf::Sprite player1RoundsSprite

  Denotes Sprite of background where number of rounds won by first player is displayed.
- sf::Sprite player2RoundsSprite

  Denotes Sprite of background where number of rounds won by second player is displayed.
- sf::Sprite gemSprite

  Denotes Sprite of gem which gets generated in the game every 5 seconds used to display gem in window.
- int velocityIterations = 10

  Denotes the iterations count of velocity in velocity phase of constraint solver in Box2D.
- int positionIterations = 10

  Denotes the iterations count of position in position phase of constraint solver in Box2D.
- float timeStep = 1.0f / 240.0f

  Timestemp for Box2D integrator.
- myListener ∗ listener

  Pointer to object listener of class type myListener of Box2D.
- std::thread worker [30]

  Worker threads initialized to 30.
- std::thread gemThread

  Thread used to generate gem in window.
- bool isClient

  Denotes Sprite of gem which gets generated in the game every 5 seconds used to display gem in window.
- bool gemExists

  Boolean variable which is true if gem exists in window, else false.
- bool isPlaying

  Boolean variable which is true until game finishes.
- bool isExiting

Boolean variable which is true while window is open, and becomes false when window is closed.

- std::mutex m

  Mutex lock used to protect and avoid simultaneous access to shared variable of first player health points and second player health points by multiple threads.
- std::mutex m1

  Mutex lock used to protect and avoid simultaneous access to shared boolean variable gemExists by multiple threads.
- Player ∗ player1

  Pointer to first player's Player object.
- Player ∗ player2

  Pointer to second player's Player object.
- std::pair< int, int > p

  Pair of integers where first paramter contains user data of body part of first player and second parameter contains user data of body part of second player which were involved in collision.
- int player1Rounds

  Denotes the number of rounds won by first player.
- int player2Rounds

  Denotes the number of rounds won by second player.
- struct timeval current_time

  structure of timeval which gives the current time in seconds and microseconds.
- struct timeval prev_time

  structure of timeval which gives the current time in seconds and microseconds.
- struct timeval current_time1

  structure of timeval which gives the current time in seconds and microseconds.
- struct timeval prev_time1

  structure of timeval which gives the current time in seconds and microseconds.
- double time_difference

  Used to check collision when time difference becomes greater than 100 milliseconds that is after every 100 milliseconds.
- double time_difference1

  Used to generate gem when time difference becomes greater than 5 seconds that is after every 5 seconds.
- bool accept =false
- bool accept1 =false

  Boolean variable containing status of connection of client to second port of server intialized to false Contains status of connection of client to server.
- sf::Clock _clock

  SFML Clock Used to display text about result of round after every round is finished for stipulated amount of time, here 3 seconds.
- sf::Text rtext

  SFML Text Text containing string "ROUND OVER" displayed after every round is finished.
- sf::Text rtext1

  SFML Text Text containing the string "first player's name wins" displayed after round is over if first player wins that round.
- sf::Text rtext2

  SFML Text Text containing the string "second player's name wins" displayed after round is over if second player wins that round.
- sf::Text rtext3

  SFML Text TText containing string "TIE" displayed if result of round is tie.
- sf::Text player1NameText

  SFML Text Text containing first player name which is displayed on window.
- sf::Text player2NameText

  SFML Text Text containing second player name which is displayed on window.
- sf::Text player1RoundsText

SFML Text Text containing number of rounds won by second player which is displayed on window.

- sf::Text player2RoundsText

  SFML Text Text containing number of rounds won by second player which is displayed on window.
- sf::Font font

  SFML Font Contains the font which is to be loaded to text to display it on window.

6.3.3 Detailed Description

Contains all basic entities required in game like window, players object, TcpListener, Send and receive sockets, Box2D world, walls,ground, sprites of all bodies to be displayed in window and functions to check collision, decrease health points, sending and receving packets from client to server and vice versa, worker threads which checks for collision and gem thread used to generate gem.

6.3.4 Constructor & Destructor Documentation

6.3.4.1 Game()

```
stickman::Game::Game (
            GameDataRef data,
            std::string s,
            bool client,
            std::string myip )
```

Constructor for game.

Parameters

data	Contains StateMachine of Game, Render Window, over which game is displayed, AssetManager of game, InputManager of Game.
s	Contains name of player.
client	Boolean which is true if its client, false for server.
myip	Contains IP over which server hosts the server used by client to connect to.

6.3.4.2 ∼Game()

```
stickman::Game::∼Game ( )
```

Destructor for game.

6.3.5 Member Function Documentation

6.3.5.1 checkcollision()

```
void stickman::Game::checkcollision ( )
```

It checks collision between body parts of first player and second player.

Here, Worker threads call the function decrease hp, for all type collisions happened simultaneously betweeen different body parts of first and second player. After collisions are handled, the worker threads are joined with main thread.

6.3.5.2 client_receive()

```
void stickman::Game::client_receive (
             float * x,
             float * y,
             float * angle,
             int * hp,
             float * gempos )
```

Used to reveive packets of information like both player's position, angle, hp and gem's position in client side from server side.

Parameters

x	It is an array consisting of x coordinates of different body parts of first and second player's position in the world.
y	It is an array consisting of y coordinates of different body parts of first and second player's position in the world.
angle	It is an array consisting of angle of different body parts of first and second player's position in the world.
hp	It is an array consisting of health points of first and second player.
gempos	It is an array consisting of x and y coordinates of gem in the window.

6.3.5.3 client_send()

```
void stickman::Game::client_send ( )
```

Used to send packets of information like the key pressed by client to move the player in client side to server side.

6.3.5.4 connect()

```
void stickman::Game::connect ( )
```

In server side, TCP listener listens to client on first and second port and on client side, it is used to connect to IP address of server and the two ports.

6.3.5.5 createGround()

```
b2Body * stickman::Game::createGround (
             b2Vec2 position,
             int angle )
```

It is used to create ground Box2D body in the Box2D world.

Parameters

position	Denotes the position of ground to be set in world.
angle	Denotes the angle of ground with respect to X-axis.

6.3.5.6 decrease_hp()

```
void stickman::Game::decrease_hp (
             int a,
             int b )
```

It is used to decrease health point of player according to collision of different body parts of each player.

Parameters

a	Denotes User Data of body part of first player in collision.
b	Denotes User Data of body part of second player in collision.

6.3.5.7 destroyBody()

```
void stickman::Game::destroyBody ( )
```

Used to destroy Box2D bodies that is head, body, right and left hand and legs of player object.

6.3.5.8 distance()

```
float stickman::Game::distance (
             int x1,
             int y1,
             int x2,
             int y2 )
```

Cacluclates distance between two points in the window.

Used to check if gem overlaps with head of any player and accordingly increase the player's health points.

Parameters

x1	X-coordinate of first point.
y1	Y-coordinate of first point.
x2	X-coordinate of second point.
y2	Y-coordinate of second point.

6.3.5.9 draw()

```
void stickman::Game::draw (
             Player * player )
```

It is used to draw different body parts like head, body, hands and legs of player in window.

Parameters

player	Takes pointer of object player to update its position.

6.3.5.10 gameLoop()

```
void stickman::Game::gameLoop ( )
```

Game loop which runs until game is finished.

Separate game loop for client and server.

6.3.5.11 generateGem()

```
void stickman::Game::generateGem ( )
```

It generates gem at random positions in the window after every 5 seconds and checks if any player has collected gem and accordingly increases its health points by 5.

It runs on separate thread until game is running.

6.3.5.12 getPlayerRounds()

```
int stickman::Game::getPlayerRounds (
             bool player )
```

It returns number of rounds won by player.

Parameters

flag	if player is true, it returns rounds won by first player, else second player.

6.3.5.13 initPlayer()

```
void stickman::Game::initPlayer (
             Player * player,
             float X,
             int offset )
```

Initialization of player object.

Used to create Box2D bodies of head, body, hands, legs and set their user data.

Parameters

player	Takes pointer of onject player whose initalization is to done.
X	Denotes inital X coordinate of player during initialization.
offset	Used to set the user data for differnt bodies of each player. For first player offset is 0 and for second player it is 1.

6.3.5.14 server_receive()

```
void stickman::Game::server_receive ( )
```

Used to receive packets of information in server side of the key pressed by client so that server can simulate it in its world.

6.3.5.15 server_send()

```
void stickman::Game::server_send ( )
```

Used to send packets of information like position of all body parts of each player and position of gem from server side to client side.

6.3.5.16 serverListen()

```
void stickman::Game::serverListen (
             bool flag )
```

TCP listener on server side listens if client has connected to any port on server's IP Address.

Parameters

flag	If flag is false, TCP listener listens on first port and if flag is true, it listens on second port.

6.3.5.17 setPlayerRounds()

```
void stickman::Game::setPlayerRounds (
             bool player,
             int rounds )
```

Sets the number of rounds won by player.

Parameters

player	if player is true, it sets rounds of first player, else second player.
rounds	Number of rounds to be set.

6.3.5.18 updatePlayer()

```
void stickman::Game::updatePlayer (
            Player * player )
```

It is used to update positions and angle of all body parts of each player after each time frame.

6.3.6 Member Data Documentation

6.3.6.1 _clock

```
sf::Clock stickman::Game::_clock
```

SFML Clock Used to display text about result of round after every round is finished for stipulated amount of time, here 3 seconds.

6.3.6.2 accept

```
bool stickman::Game::accept =false
```

6.3.6.3 accept1

```
bool stickman::Game::accept1 =false
```

Boolean variable containing status of connection of client to second port of server intialized to false Contains status of connection of client to server.

It becomes true if client connects to second port and TCP listener on server side accepts it.

6.3.6.4 current_time

```
struct timeval stickman::Game::current_time
```

structure of timeval which gives the current time in seconds and microseconds.

Used to store current time to check collision.

6.3.6.5 current_time1

```
struct timeval stickman::Game::current_time1
```

structure of timeval which gives the current time in seconds and microseconds.

Used to store current time to generate gem.

6.3.6.6 font

```
sf::Font stickman::Game::font
```

SFML Font Contains the font which is to be loaded to text to display it on window.

6.3.6.7 gemExists

```
bool stickman::Game::gemExists
```

Boolean variable which is true if gem exists in window, else false.

Used to display the gem in window if the boolean is true.

6.3.6.8 gemSprite

```
sf::Sprite stickman::Game::gemSprite
```

Denotes Sprite of gem which gets generated in the game every 5 seconds used to display gem in window.

6.3.6.9 gemTexture

```
sf::Texture stickman::Game::gemTexture
```

Denotes texture of gem which gets generated in the game every 5 seconds.

6.3.6.10 gemThread

```
std::thread stickman::Game::gemThread
```

Thread used to generate gem in window.

Thread generates gem in window after every 5 seconds and checks if gem has been consumed by player or not. Thread runs until game is finished and joins with the main thread.

6.3.6.11 ground

```
b2Body* stickman::Game::ground
```

It is a Box2D body object pointer which is denoting the ground situated in world.

6.3.6.12 groundSprite

```
sf::Sprite stickman::Game::groundSprite
```

Denotes Sprite of ground used to display ground in window.

6.3.6.13 groundTexture

```
sf::Texture stickman::Game::groundTexture
```

Denotes texture of ground where images of ground is to loaded.

6.3.6.14 groundUserData

```
int stickman::Game::groundUserData
```

Contains the userData of ground.

```
Used to detect rcollision of a body part of player with ground.
```

6.3.6.15 ip

```
sf::IpAddress stickman::Game::ip
```

Denotes the IP Address over which server hosts.

6.3.6.16 isClient

```
bool stickman::Game::isClient
```

Denotes Sprite of gem which gets generated in the game every 5 seconds used to display gem in window.

6.3.6.17 isExiting

```
bool stickman::Game::isExiting
```

Boolean variable which is true while window is open, and becomes false when window is closed.

Used to run game loop until window is closed or either of player wins the game.

6.3.6.18 isPlaying

```
bool stickman::Game::isPlaying
```

Boolean variable which is true until game finishes.

Used to generate and display gem until the boolean is true.

6.3.6.19 listener

```
myListener* stickman::Game::listener
```

Pointer to object listener of class type myListener of Box2D.

It is used to detect collisions between bodies and give the information about the point of contact of colliding bodies and impulse applied during collision.

6.3.6.20 listenSocket

```
sf::TcpSocket stickman::Game::listenSocket
```

It is a TCP Socket used for sending packets from client side and receving packets in client side.

6.3.6.21 m

```
std::mutex stickman::Game::m
```

Mutex lock used to protect and avoid simultaneous access to shared variable of first player health points and second player health points by multiple threads.

6.3.6.22 m1

```
std::mutex stickman::Game::m1
```

Mutex lock used to protect and avoid simultaneous access to shared boolean variable gemExists by multiple threads.

6.3.6.23 myip

```
std::string stickman::Game::myip
```

It is a string containing IP Address of server.

6.3.6.24 p

```
std::pair<int,int> stickman::Game::p
```

Pair of integers where first paramter contains user data of body part of first player and second parameter contains user data of body part of second player which were involved in collision.

6.3.6.25 player1

```
Player* stickman::Game::player1
```

Pointer to first player's Player object.

It contains information about everything about first player.

Pointer is needed to move and rotate the player, set its health points.

6.3.6.26 player1NameText

```
sf::Text stickman::Game::player1NameText
```

SFML Text Text containing first player name which is displayed on window.

6.3.6.27 player1Rounds

```
int stickman::Game::player1Rounds
```

Denotes the number of rounds won by first player.

6.3.6.28 player1RoundsSprite

```
sf::Sprite stickman::Game::player1RoundsSprite
```

Denotes Sprite of background where number of rounds won by first player is displayed.

6.3.6.29 player1RoundsText

```
sf:: Text stickman::Game::player1RoundsText
```

SFML Text Text containing number of rounds won by second player which is displayed on window.

6.3.6.30 player2

```
Player* stickman::Game::player2
```

Pointer to second player's Player object.

It contains information about everything about second player.

Pointer is needed to move and rotate the player, set its health points.

6.3.6.31 player2NameText

```
sf:: Text stickman::Game::player2NameText
```

SFML Text Text containing second player name which is displayed on window.

6.3.6.32 player2Rounds

```
int stickman::Game::player2Rounds
```

Denotes the number of rounds won by second player.

6.3.6.33 player2RoundsSprite

```
sf::Sprite stickman::Game::player2RoundsSprite
```

Denotes Sprite of background where number of rounds won by second player is displayed.

6.3.6.34 player2RoundsText

```
sf:: Text stickman::Game::player2RoundsText
```

SFML Text Text containing number of rounds won by second player which is displayed on window.

6.3.6.35 positionIterations

```
int stickman::Game::positionIterations = 10
```

Denotes the iterations count of position in position phase of constraint solver in Box2D.

In the position phase the solver adjusts the positions of the bodies to reduce overlap and joint detachment.

6.3.6.36 prev_time

```
struct timeval stickman::Game::prev_time
```

structure of timeval which gives the current time in seconds and microseconds.

Used to store previous time to check collision.

6.3.6.37 prev_time1

```
struct timeval stickman::Game::prev_time1
```

structure of timeval which gives the current time in seconds and microseconds.

Used to store previous time to generate gem.

6.3.6.38 roundTexture

```
sf::Texture stickman::Game::roundTexture
```

Denotes texture of background where number of rounds won by each player is displayed.

6.3.6.39 rtext

```
sf::Text stickman::Game::rtext
```

SFML Text Text containing string "ROUND OVER" displayed after every round is finished.

6.3.6.40 rtext1

```
sf::Text stickman::Game::rtext1
```

SFML Text Text containing the string "first player's name wins" displayed after round is over if first player wins that round.

6.3.6.41 rtext2

```
sf::Text stickman::Game::rtext2
```

SFML Text Text containing the string "second player's name wins" displayed after round is over if second player wins that round.

6.3.6.42 rtext3

```
sf::Text stickman::Game::rtext3
```

SFML Text TText containing string "TIE" displayed if result of round is tie.

6.3.6.43 sendSocket

```
sf::TcpSocket stickman::Game::sendSocket
```

It is a TCP Socket used for sending packets from server side and receiving packets in client side.

6.3.6.44 tcplistener

```
sf::TcpListener stickman::Game::tcplistener
```

Basically, it is a TCP listener listens to a particular port and accepts if client connects.

6.3.6.45 tcplistener1

```
sf::TcpListener stickman::Game::tcplistener1
```

It is a TCP listener whichlistens to a particular port and accepts if client connects.

6.3.6.46 time_difference

```
double stickman::Game::time_difference
```

Used to check collision when time difference becomes greater than 100 milliseconds that is after every 100 milliseconds.

6.3.6.47 time_difference1

```
double stickman::Game::time_difference1
```

Used to generate gem when time difference becomes greater than 5 seconds that is after every 5 seconds.

6.3.6.48 timeStep

```
float stickman::Game::timeStep = 1.0f / 240.0f
```

Timestemp for Box2D integrator.

6.3.6.49 velocityIterations

```
int stickman::Game::velocityIterations = 10
```

Denotes the iterations count of velocity in velocity phase of constraint solver in Box2D.

In the velocity phase the solver computes the impulses necessary for the bodies to move correctly.

6.3.6.50 wall1

```
b2Body* stickman::Game::wall1
```

It is a Box2D body object pointer which is denoting the upper wall situated in world.

6.3.6.51 wall1Sprite

```
sf::Sprite stickman::Game::wall1Sprite
```

Denotes Sprite of upper wall used to display upper wall in window.

6.3.6.52 wall1Texture

```
sf::Texture stickman::Game::wall1Texture
```

Denotes texture of upper wall where image of ground is to loaded.

6.3.6.53 wall2

```
b2Body* stickman::Game::wall2
```

It is a Box2D body object pointer which is denoting the left most wall situated in world.

6.3.6.54 wall2Sprite

```
sf::Sprite stickman::Game::wall2Sprite
```

Denotes Sprite of lefmost wall used to display leftmost wall in window.

6.3.6.55 wall2Texture

```
sf::Texture stickman::Game::wall2Texture
```

Denotes texture of left most wall where image of wall is to loaded.

6.3.6.56 wall3

```
b2Body* stickman::Game::wall3
```

It is a Box2D body object pointer which is denoting the right most wall situated in world.

6.3.6.57 wall3Sprite

```
sf::Sprite stickman::Game::wall3Sprite
```

Denotes Sprite of rightmost wall used to display rightmost wall in window.

6.3.6.58 wall3Texture

```
sf::Texture stickman::Game::wall3Texture
```

Denotes texture of right most wall where image of wall is to loaded.

6.3.6.59 window

```
sf::RenderWindow* stickman::Game::window
```

The main SFML window of type RenderWindow over which game is displayed.

6.3.6.60 worker

```
std::thread stickman::Game::worker[30]
```

Worker threads initialized to 30.

Used to detect collision between multiple body parts of two players or body parts of a player with ground or walls and modify health points accordingly simultaneously.

6.3.6.61 world

```
b2World* stickman::Game::world
```

It is Box2D world object pointer where all the bodies of player reside.

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

- game.h
- game.cpp

6.4 stickman::Game2 Class Reference

Class for game which initializes the different properties related to the game like resolution and so on.

```
#include <Game.hpp>
```

6.4.1 *

Public Member Functions

- Game2 (int width, int height, std::string title)

  Creates the window with the given resolution,framerate and title while also adding the first state.

6.4.2 Detailed Description

Class for game which initializes the different properties related to the game like resolution and so on.

6.4.3 Constructor & Destructor Documentation

6.4.3.1 Game2()

```
stickman::Game2::Game2 (
              int width,
              int height,
              std::string title )
```

Creates the window with the given resolution,framerate and title while also adding the first state.

Parameters

width	The width of screen
height	The height of screen
title	The title of window screen

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

- Game.hpp
- Game.cpp

6.5 stickman::GameData Struct Reference

This contains the objects required by the game as the whole like the state machine which switches states and different managers to make loading different things easier.

```
#include <Game.hpp>
```

6.5.1 *

Public Attributes

- StateMachine machine
- sf::RenderWindow window
- AssetManager assets
- InputManager input

6.5.2 Detailed Description

This contains the objects required by the game as the whole like the state machine which switches states and different managers to make loading different things easier.

6.5.3 Member Data Documentation

6.5.3.1 assets

```
AssetManager stickman::GameData::assets
```

6.5.3.2 input

```
InputManager stickman::GameData::input
```

6.5.3.3 machine

```
StateMachine stickman::GameData::machine
```

6.5.3.4 window

```
sf::RenderWindow stickman::GameData::window
```

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

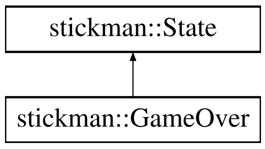
- Game.hpp

6.6 stickman::GameOver Class Reference

Class for game over state.

```
#include <GameOver.h>
```

Inheritance diagram for stickman::GameOver:



6.6.1 *

Public Member Functions

- GameOver (GameDataRef data, std::string name, int result)

  Constructs the object.
- void Init ()

  Virtual function init that may be overloaded which runs at the start of the state.
- void HandleInput ()

  Virtual function HandleInput that may be overloaded which may be used to handle some input.
- void Update (float dt)

  Virtual function Update which may be overloaded which may be used to update game logic.
- void Draw (float dt)

  Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

6.6.2 *

Public Attributes

- sf::Font font

  Stores a font.
- sf::Text gtext

  Stores text to be displayed.
- sf::Text gtext1

6.6.3 Detailed Description

Class for game over state.

6.6.4 Constructor & Destructor Documentation

6.6.4.1 GameOver()

```
stickman::GameOver::GameOver (
             GameDataRef data,
             std::string name,
             int result )
```

Constructs the object.

Parameters

data	The data which contains information about the game
name	Stores the name of winner
result	Stores the result if there is a win / tie.

6.6.5 Member Function Documentation

6.6.5.1 Draw()

```
void stickman::GameOver::Draw (
             float dt )  [virtual]
```

Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.6.5.2 HandleInput()

```
void stickman::GameOver::HandleInput ( )  [virtual]
```

Virtual function HandleInput that may be overloaded which may be used to handle some input.

Implements stickman::State.

6.6.5.3 Init()

```
void stickman::GameOver::Init ( )  [virtual]
```

Virtual function init that may be overloaded which runs at the start of the state.

Implements stickman::State.

6.6.5.4 Update()

```
void stickman::GameOver::Update (
             float dt )  [virtual]
```

Virtual function Update which may be overloaded which may be used to update game logic.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.6.6 Member Data Documentation

6.6.6.1 font

```
sf::Font stickman::GameOver::font
```

Stores a font.

6.6.6.2 gtext

```
sf::Text stickman::GameOver::gtext
```

Stores text to be displayed.

6.6.6.3 gtext1

```
sf::Text stickman::GameOver::gtext1
```

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

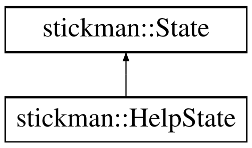
- GameOver.h
- GameOver.cpp

6.7 stickman::HelpState Class Reference

Class for help state.

```
#include <HelpState.hpp>
```

Inheritance diagram for stickman::HelpState:



6.7.1 *

Public Member Functions

- HelpState (GameDataRef data)

Constructs the object.
- void Init ()

  Virtual function init that may be overloaded which runs at the start of the state.
- void HandleInput ()

  Virtual function HandleInput that may be overloaded which may be used to handle some input.
- void Update (float dt)

  Virtual function Update which may be overloaded which may be used to update game logic.
- void Draw (float dt)

  Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.
- void setText (sf::Text text, int xPos, int yPos, int size, string s)

  Sets the text onto a particular position.

6.7.2 *

Public Attributes

- sf::Font font

  Stores a font.
- sf::Text infoText

  Stores text to be displayed.
- sf::Text rulesText
- sf::Text rule1Text
- sf::Text rule2Text
- sf::Text rule3Text
- sf::Sprite backSprite

  Sprite which stores the back button sprite.

6.7.3 Detailed Description

Class for help state.

6.7.4 Constructor & Destructor Documentation

6.7.4.1 HelpState()

```
stickman::HelpState::HelpState (
            GameDataRef data )
```

Constructs the object.

Parameters

data	The data which contains information about the game

6.7.5 Member Function Documentation

6.7.5.1 Draw()

```
void stickman::HelpState::Draw (
             float dt )  [virtual]
```

Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.7.5.2 HandleInput()

```
void stickman::HelpState::HandleInput ( )  [virtual]
```

Virtual function HandleInput that may be overloaded which may be used to handle some input.

Implements stickman::State.

6.7.5.3 Init()

```
void stickman::HelpState::Init ( )  [virtual]
```

Virtual function init that may be overloaded which runs at the start of the state.

Implements stickman::State.

6.7.5.4 setText()

```
void stickman::HelpState::setText (
             sf::Text text,
             int xPos,
             int yPos,
             int size,
             string s )
```

Sets the text onto a particular position.

Parameters

text	The text
xPos	The x position
yPos	The y position
size	The size
s	Text to be set

6.7.5.5 Update()

```
void stickman::HelpState::Update (
            float dt )  [virtual]
```

Virtual function Update which may be overloaded which may be used to update game logic.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.7.6 Member Data Documentation

6.7.6.1 backSprite

```
sf::Sprite stickman::HelpState::backSprite
```

Sprite which stores the back button sprite.

6.7.6.2 font

```
sf::Font stickman::HelpState::font
```

Stores a font.

6.7.6.3 infoText

```
sf::Text stickman::HelpState::infoText
```

Stores text to be displayed.

6.7.6.4 rule1Text

```
sf::Text stickman::HelpState::rule1Text
```

6.7.6.5 rule2Text

```
sf::Text stickman::HelpState::rule2Text
```

6.7.6.6 rule3Text

```
sf::Text stickman::HelpState::rule3Text
```

6.7.6.7 rulesText

```
sf::Text stickman::HelpState::rulesText
```

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

- HelpState.hpp
- HelpState.cpp

6.8 stickman::InputManager Class Reference

Class for input manager.

```
#include <InputManager.hpp>
```

6.8.1 *

Public Member Functions

- InputManager ()

  Constructs the object.
- ∼InputManager ()

  Destroys the object.
- bool IsSpriteClicked (sf::Sprite object, sf::Mouse::Button button, sf::RenderWindow &window)

  Determines if sprite is clicked.
- sf::Vector2i GetMousePosition (sf::RenderWindow &window)

  Gets the mouse position.

6.8.2 Detailed Description

Class for input manager.

6.8.3 Constructor & Destructor Documentation

6.8.3.1 InputManager()

```
stickman::InputManager::InputManager ( )  [inline]
```

Constructs the object.

6.8.3.2 ∼InputManager()

```
stickman::InputManager::∼InputManager ( )  [inline]
```

Destroys the object.

6.8.4 Member Function Documentation

6.8.4.1 GetMousePosition()

```
sf::Vector2i stickman::InputManager::GetMousePosition (
              sf::RenderWindow & window )
```

Gets the mouse position.

Parameters

window	The window on which the game is running

Returns

The mouse position.

6.8.4.2 IsSpriteClicked()

```
bool stickman::InputManager::IsSpriteClicked (
             sf::Sprite object,
             sf::Mouse::Button button,
             sf::RenderWindow & window )
```

Determines if sprite is clicked.

Parameters

in	*object*	The sprite with which we are checking
in	*button*	The button which should be pressed
	window	The window on which the game is running

Returns

True if sprite clicked, False otherwise.

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

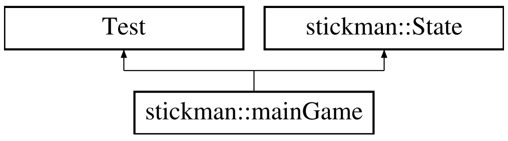
- InputManager.hpp
- InputManager.cpp

6.9 stickman::mainGame Struct Reference

Class for main game.

`#include <mainGame.hpp>`

Inheritance diagram for stickman::mainGame:



6.9.1 *

Public Member Functions

- mainGame (GameDataRef data, string s, bool client, string ip)

Constructs the object.

- void Init ()

  Virtual function init that may be overloaded which runs at the start of the state.
- void HandleInput ()

  Virtual function HandleInput that may be overloaded which may be used to handle some input.
- void Update (float dt)

  Virtual function Update which may be overloaded which may be used to update game logic.
- void Draw (float dt)

  Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.
- mainGame ()

  Constructor of struct.
- virtual ∼mainGame ()

  Destroys the object.

6.9.2 *

Public Attributes

- GameDataRef ∗ data
- Game ∗ temp
- std::string t1 ="a"
- bool t2 =true

6.9.3 Detailed Description

Class for main game.

A struct passed for testing game functions.

6.9.4 Constructor & Destructor Documentation

6.9.4.1 mainGame() [1/2]

```
stickman::mainGame::mainGame (
             GameDataRef data,
             string s,
             bool client,
             string ip )
```

Constructs the object.

Parameters

data	The data which contains information about the game
s	Stores the name of player
client	Stores the information whether the system is client/server
ip	IP to be connected

6.9.4.2 mainGame() [2/2]

```
stickman::mainGame::mainGame ( )  [inline]
```

Constructor of struct.

6.9.4.3 ∼mainGame()

```
virtual stickman::mainGame::∼mainGame ( )  [inline], [virtual]
```

Destroys the object.

6.9.5 Member Function Documentation

6.9.5.1 Draw()

```
void stickman::mainGame::Draw (
             float dt )  [virtual]
```

Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.9.5.2 HandleInput()

```
void stickman::mainGame::HandleInput ( )  [virtual]
```

Virtual function HandleInput that may be overloaded which may be used to handle some input.

Implements stickman::State.

6.9.5.3 Init()

```
void stickman::mainGame::Init ( )  [virtual]
```

Virtual function init that may be overloaded which runs at the start of the state.

Implements stickman::State.

6.9.5.4 Update()

```
void stickman::mainGame::Update (
             float dt )  [virtual]
```

Virtual function Update which may be overloaded which may be used to update game logic.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.9.6 Member Data Documentation

6.9.6.1 data

```
GameDataRef* stickman::mainGame::data
```

6.9.6.2 t1

```
std::string stickman::mainGame::t1 ="a"
```

6.9.6.3 t2

```
bool stickman::mainGame::t2 =true
```

6.9.6.4 temp

```
Game* stickman::mainGame::temp
```

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

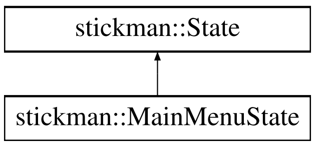
- mainGame.hpp
- tests.hpp
- mainGame.cpp

6.10 stickman::MainMenuState Class Reference

Class for main menu state.

```
#include <MainMenuState.hpp>
```

Inheritance diagram for stickman::MainMenuState:



6.10.1 *

Public Member Functions

- MainMenuState (GameDataRef data, string s, bool client)

    Constructs the object.
- void Init ()

    Virtual function init that may be overloaded which runs at the start of the state.
- void HandleInput ()

    Virtual function HandleInput that may be overloaded which may be used to handle some input.
- void Update (float dt)

    Virtual function Update which may be overloaded which may be used to update game logic.
- void Draw (float dt)

    Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

6.10.2 *

Public Attributes

- sf::Text playerText

    SFML text to be displayed as input is taken.
- sf::Font font

    SFML font to be loaded.
- std::string playerInput

    String containing the ip to be connected.
- sf::Text text1

    SFML Text to be diplayed.
- sf::Text text2

    SFML Text to be diplayed.

6.10.3 Detailed Description

Class for main menu state.

6.10.4 Constructor & Destructor Documentation

6.10.4.1 MainMenuState()

```
stickman::MainMenuState::MainMenuState (
            GameDataRef data,
            string s,
            bool client )
```

Constructs the object.

Parameters

data	Takes the data of the game from the previous state
s	Stores the name of player
client	Stores the information whether the system is client/server

6.10.5 Member Function Documentation

6.10.5.1 Draw()

```
void stickman::MainMenuState::Draw (
            float dt )  [virtual]
```

Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.10.5.2 HandleInput()

```
void stickman::MainMenuState::HandleInput ( )  [virtual]
```

Virtual function HandleInput that may be overloaded which may be used to handle some input.

Implements stickman::State.

6.10.5.3 Init()

```
void stickman::MainMenuState::Init ( )  [virtual]
```

Virtual function init that may be overloaded which runs at the start of the state.

Implements stickman::State.

6.10.5.4 Update()

```
void stickman::MainMenuState::Update (
             float dt )  [virtual]
```

Virtual function Update which may be overloaded which may be used to update game logic.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.10.6 Member Data Documentation

6.10.6.1 font

```
sf::Font stickman::MainMenuState::font
```

SFML font to be loaded.

6.10.6.2 playerInput

```
std::string stickman::MainMenuState::playerInput
```

String containing the ip to be connected.

6.10.6.3 playerText

```
sf::Text stickman::MainMenuState::playerText
```

SFML text to be displayed as input is taken.

6.10.6.4 text1

```
sf::Text stickman::MainMenuState::text1
```

SFML Text to be diplayed.

6.10.6.5 text2

```
sf::Text stickman::MainMenuState::text2
```

SFML Text to be diplayed.

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

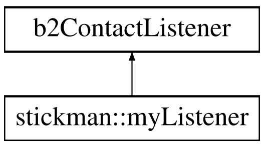
- MainMenuState.hpp
- MainMenuState.cpp

6.11 stickman::myListener Class Reference

Listens to collision between any two objects in Box2D world.

```
#include <myListener.h>
```

Inheritance diagram for stickman::myListener:



6.11.1 *

Public Member Functions

- myListener ()

  Constructor for listener.
- ∼myListener ()

  Destructor for listener.
- void BeginContact (b2Contact ∗contact)

  Called when an object starts collision with other object.

6.11.2 *

Public Attributes

- std::queue< std::pair< int, int > > Queue

  Stores the id of two body parts which collided.

6.11.3 Detailed Description

Listens to collision between any two objects in Box2D world.

```
Subclass of b2ContactListener which implements the virtual method BeginContact

Contains the queue in which id of body part is stored which is further processed to decrease health point
```

6.11.4 Constructor & Destructor Documentation

6.11.4.1 myListener()

```
stickman::myListener::myListener ( )
```

Constructor for listener.

6.11.4.2 ∼myListener()

```
stickman::myListener::∼myListener ( )
```

Destructor for listener.

6.11.5 Member Function Documentation

6.11.5.1 BeginContact()

```
void stickman::myListener::BeginContact (
            b2Contact * contact )
```

Called when an object starts collision with other object.

Parameters

contact	Stores the contact information of two objects in Box2D world.

6.11.6 Member Data Documentation

6.11.6.1 Queue

```
std::queue< std::pair<int,int> > stickman::myListener::Queue
```

Stores the id of two body parts which collided.

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

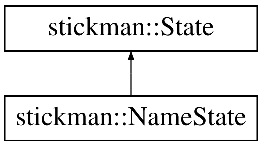
- myListener.h
- myListener.cpp

6.12 stickman::NameState Class Reference

Class for name state which takes the name of player and gives the option of chosing whether to host a server/ join a server.

```
#include <name.h>
```

Inheritance diagram for stickman::NameState:



6.12.1 *

Public Member Functions

- NameState (GameDataRef data)

  Constructs the object.
- void Init ()

  Virtual function init that may be overloaded which runs at the start of the state.
- void HandleInput ()

  Virtual function HandleInput that may be overloaded which may be used to handle some input.
- void Draw (float dt)

  Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.
- void Update (float dt)

  Virtual function Update which may be overloaded which may be used to update game logic.

6.12.2 *

Public Attributes

- sf::Text enterName
- sf::Texture welcomeTexture
- sf::Sprite welcomeSprite
- sf::Text playerText
- sf::Font font
- std::string playerInput

6.12.3 Detailed Description

Class for name state which takes the name of player and gives the option of chosing whether to host a server/ join a server.

6.12.4 Constructor & Destructor Documentation

6.12.4.1 NameState()

```
stickman::NameState::NameState (
            GameDataRef data )
```

Constructs the object.

Parameters

data	Takes the data of the game from the previous state

6.12.5 Member Function Documentation

6.12.5.1 Draw()

```
void stickman::NameState::Draw (
            float dt )  [virtual]
```

Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.12.5.2 HandleInput()

```
void stickman::NameState::HandleInput ( )  [virtual]
```

Virtual function HandleInput that may be overloaded which may be used to handle some input.

Implements stickman::State.

6.12.5.3 Init()

```
void stickman::NameState::Init ( )  [virtual]
```

Virtual function init that may be overloaded which runs at the start of the state.

Implements stickman::State.

6.12.5.4 Update()

```
void stickman::NameState::Update (
            float dt )  [virtual]
```

Virtual function Update which may be overloaded which may be used to update game logic.

Parameters

dt	The difference in frames to syncronise with framerate

Implements stickman::State.

6.12.6 Member Data Documentation

6.12.6.1 enterName

```
sf::Text stickman::NameState::enterName
```

6.12.6.2 font

```
sf::Font stickman::NameState::font
```

6.12.6.3 playerInput

```
std::string stickman::NameState::playerInput
```

6.12.6.4 playerText

```
sf::Text stickman::NameState::playerText
```

6.12.6.5 welcomeSprite

```
sf::Sprite stickman::NameState::welcomeSprite
```

6.12.6.6 welcomeTexture

```
sf::Texture stickman::NameState::welcomeTexture
```

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

- name.h
- name.cpp

6.13 stickman::Player Class Reference

Contains all the information about the player.

`#include <player.h>`

6.13.1 *

Public Member Functions

- Player ()

  Constructor for player.
- ∼Player ()

  Destructor for player.
- b2Body ∗ createhead (b2World ∗world, b2Vec2 position, bool isStatic, float radius, float restitution, float density)

  Creates a circular Box2D object for head of player.
- b2Body ∗ createbody (b2World ∗world, b2Vec2 position, bool isStatic, float length, float width, float restitution, float density)

  Creates a rectangular Box2D object for each body part(body, hands, legs) of player.
- b2RevoluteJoint ∗ createRevoluteJoint (b2World ∗world, b2Body ∗body1, b2Body ∗body2, b2Vec2 anchor↩ Point1, b2Vec2 anchorPoint2, float lowerLimit, float upperLimit)

  Revolute joints can be think of as hinge which allows rotation of body parts.
- void setHealth (int health)

  Sets the health point of player.
- void setName (std::string name)

  Sets the name of player.
- std::string getName ()

  Returns the name of player.
- int getHealth ()

  Returns the health of player.
- void init (bool firstPlayer)

  Initialises the textures and sprites of a player.

6.13.2 *

Public Attributes

- int health

  Health of player Contains the current health point of player.
- b2Body ∗ head

  Head of player Contains the Box2D pointer of the object denoting player's head.
- int headUserData

  Contains the userData of head.
- sf::Texture headTexture

  SFML Texture to be loaded for head of player.
- sf::Sprite headSprite

  SFML Sprite to be displayed for head of player.
- b2Body ∗ body

  Body of player Contains the Box2D pointer of the object denoting player's body.

- int bodyUserData

  Contains the userData of body.
- sf::Texture bodyTexture

  SFML Texture to be loaded for body of player.
- sf::Sprite bodySprite

  SFML Sprite to be displayed for body of player.
- b2Body ∗ left_hand

  Left Hand of player Contains the Box2D pointer of the object denoting player's left hand.
- int left_handUserData

  Contains the userData of player's left hand.
- sf::Texture handTexture

  SFML Texture to be loaded for both the hands of player.
- sf::Sprite left_handSprite

  SFML Sprite to be displayed for left hand of player.
- b2Body ∗ right_hand

  Right Hand of player Contains the Box2D pointer of the object denoting player's right hand.
- int right_handUserData

  Contains the userData of player's right hand.
- sf::Sprite right_handSprite

  SFML Sprite to be displayed for right hand of player.
- b2Body ∗ left_leg

  Left Leg of player Contains the Box2D pointer of the object denoting player's left leg.
- int left_legUserData

  Contains the userData of player's left leg.
- sf::Texture legTexture

  SFML Texture to be loaded for both the legs of player.
- sf::Sprite left_legSprite

  SFML Sprite to be displayed for left leg of player.
- b2Body ∗ right_leg

  Right Leg of player Contains the Box2D pointer of the object denoting player's right leg.
- int right_legUserData

  Contains the userData of player's right leg.
- sf::Sprite right_legSprite

  SFML Sprite to be displayed for right leg of player.
- b2RevoluteJoint ∗ headJoint

  Represents Revolute Joint between head and body of player.
- b2RevoluteJoint ∗ right_handJoint

  Represents Revolute Joint between right hand and body of player.
- b2RevoluteJoint ∗ left_handJoint

  Represents Revolute Joint between left hand and body of player.
- b2RevoluteJoint ∗ right_legJoint

  Represents Revolute Joint between right leg and body of player.
- b2RevoluteJoint ∗ left_legJoint

  Represents Revolute Joint between left leg and body of player.
- std::string name

  Name of player Stores the name of player.

6.13.3 Detailed Description

Contains all the information about the player.

```
Contains information about the player including its name, health point, textures, sprites and Box2D body
pointers along with some joints.
```

6.13.4 Constructor & Destructor Documentation

6.13.4.1 Player()

```
stickman::Player::Player ( )
```

Constructor for player.

6.13.4.2 ∼Player()

```
stickman::Player::∼Player ( )
```

Destructor for player.

6.13.5 Member Function Documentation

6.13.5.1 createbody()

```
b2Body * stickman::Player::createbody (
             b2World * world,
             b2Vec2 position,
             bool isStatic,
             float length,
             float width,
             float restitution,
             float density )
```

Creates a rectangular Box2D object for each body part(body, hands, legs) of player.

Parameters

world	Box2D world in which player is initialised.
position	Contains the position where body part is to be created.
isStatic	Denotes whether the object is static or dynamic.
length	Contains the length of body part of player.
length	Contains the width of body part of player.
restitution	Contains the coefficient of restituion of body part of player.
density	Stores the density of body part of player.

Returns

Box2D object pointer of body part.

6.13.5.2 createhead()

```
b2Body * stickman::Player::createhead (
            b2World * world,
            b2Vec2 position,
            bool isStatic,
            float radius,
            float restitution,
            float density )
```

Creates a circular Box2D object for head of player.

Parameters

world	Box2D world in which player is initialised.
position	Contains the position where head is to be created.
isStatic	Denotes whether the object is static or dynamic.
radius	Contains the radius of head of player.
restitution	Contains the coefficient of restituion for head of player.
density	Stores the density of head of player.

Returns

Box2D object pointer of head.

6.13.5.3 createRevoluteJoint()

```
b2RevoluteJoint * stickman::Player::createRevoluteJoint (
            b2World * world,
            b2Body * body1,
            b2Body * body2,
            b2Vec2 anchorPoint1,
            b2Vec2 anchorPoint2,
            float lowerLimit,
            float upperLimit )
```

Revolute joints can be think of as hinge which allows rotation of body parts.

Creates a Revolute joint between two body parts.

Parameters

world	Box2D world in which player is initialised.
body1	Contains the Box2D object pointer denoting the first body part in joint.
body2	Contains the Box2D object pointer denoting the second body part in joint.
anchorPoint1	Contains the local position in first body where joint has to be initialised.
anchorPoint2	Contains the local position in second body where joint has to be initialised.
lowerLimit	Stores the lower limit of angle of rotation for the Revolute joint.
upperLimit	Stores the upper limit of angle of rotation for the Revolute joint.

Returns

Box2D Revolute joint pointer denoting the joint.

6.13.5.4 getHealth()

```
int stickman::Player::getHealth ( )
```

Returns the health of player.

Returns

Health point of player.

6.13.5.5 getName()

```
std::string stickman::Player::getName ( )
```

Returns the name of player.

Returns

Name of player.

6.13.5.6 init()

```
void stickman::Player::init (
             bool firstPlayer )
```

Initialises the textures and sprites of a player.

Parameters

firstPlayer	Stores the information whether the player is first player or second.

6.13.5.7 setHealth()

```
void stickman::Player::setHealth (
             int health )
```

Sets the health point of player.

Parameters

health	Stores the health to be set.

6.13.5.8 setName()

```
void stickman::Player::setName (
             std::string name )
```

Sets the name of player.

Parameters

name	Stores the name to be set.

6.13.6 Member Data Documentation

6.13.6.1 body

```
b2Body* stickman::Player::body
```

Body of player Contains the Box2D pointer of the object denoting player's body.

6.13.6.2 bodySprite

```
sf::Sprite stickman::Player::bodySprite
```

SFML Sprite to be displayed for body of player.

6.13.6.3 bodyTexture

```
sf::Texture stickman::Player::bodyTexture
```

SFML Texture to be loaded for body of player.

6.13.6.4 bodyUserData

```
int stickman::Player::bodyUserData
```

Contains the userData of body.

Used to detect body in collision.

6.13.6.5 handTexture

```
sf::Texture stickman::Player::handTexture
```

SFML Texture to be loaded for both the hands of player.

6.13.6.6 head

```
b2Body* stickman::Player::head
```

Head of player Contains the Box2D pointer of the object denoting player's head.

6.13.6.7 headJoint

```
b2RevoluteJoint* stickman::Player::headJoint
```

Represents Revolute Joint between head and body of player.

6.13.6.8 headSprite

```
sf::Sprite stickman::Player::headSprite
```

SFML Sprite to be displayed for head of player.

6.13.6.9 headTexture

```
sf::Texture stickman::Player::headTexture
```

SFML Texture to be loaded for head of player.

6.13.6.10 headUserData

```
int stickman::Player::headUserData
```

Contains the userData of head.

Used to detect head in collision of two body parts.

6.13.6.11 health

```
int stickman::Player::health
```

Health of player Contains the current health point of player.

6.13.6.12 left_hand

```
b2Body* stickman::Player::left_hand
```

Left Hand of player Contains the Box2D pointer of the object denoting player's left hand.

6.13.6.13 left_handJoint

```
b2RevoluteJoint* stickman::Player::left_handJoint
```

Represents Revolute Joint between left hand and body of player.

6.13.6.14 left_handSprite

```
sf::Sprite stickman::Player::left_handSprite
```

SFML Sprite to be displayed for left hand of player.

6.13.6.15 left_handUserData

```
int stickman::Player::left_handUserData
```

Contains the userData of player's left hand.

Used to detect left hand in collision

6.13.6.16 left_leg

```
b2Body* stickman::Player::left_leg
```

Left Leg of player Contains the Box2D pointer of the object denoting player's left leg.

6.13.6.17 left_legJoint

```
b2RevoluteJoint* stickman::Player::left_legJoint
```

Represents Revolute Joint between left leg and body of player.

6.13.6.18 left_legSprite

```
sf::Sprite stickman::Player::left_legSprite
```

SFML Sprite to be displayed for left leg of player.

6.13.6.19 left_legUserData

```
int stickman::Player::left_legUserData
```

Contains the userData of player's left leg.

Used to detect left leg in collision of two body parts.

6.13.6.20 legTexture

```
sf::Texture stickman::Player::legTexture
```

SFML Texture to be loaded for both the legs of player.

6.13.6.21 name

```
std::string stickman::Player::name
```

Name of player Stores the name of player.

6.13.6.22 right_hand

```
b2Body* stickman::Player::right_hand
```

Right Hand of player Contains the Box2D pointer of the object denoting player's right hand.

6.13.6.23 right_handJoint

```
b2RevoluteJoint* stickman::Player::right_handJoint
```

Represents Revolute Joint between right hand and body of player.

6.13.6.24 right_handSprite

```
sf::Sprite stickman::Player::right_handSprite
```

SFML Sprite to be displayed for right hand of player.

6.13.6.25 right_handUserData

```
int stickman::Player::right_handUserData
```

Contains the userData of player's right hand.

Used to detect right hand in collision of two body parts.

6.13.6.26 right_leg

```
b2Body* stickman::Player::right_leg
```

Right Leg of player Contains the Box2D pointer of the object denoting player's right leg.

6.13.6.27 right_legJoint

```
b2RevoluteJoint* stickman::Player::right_legJoint
```

Represents Revolute Joint between right leg and body of player.

6.13.6.28 right_legSprite

```
sf::Sprite stickman::Player::right_legSprite
```

SFML Sprite to be displayed for right leg of player.

6.13.6.29 right_legUserData

```
int stickman::Player::right_legUserData
```

Contains the userData of player's right leg.

Used to detect right leg in collision of two body parts.

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

- player.h
- player.cpp

6.14 stickman::playerdata Struct Reference

Struct for testing player data.

```
#include <tests.hpp>
```

6.14.1 *

Public Member Functions

- playerdata (int a1, int a2, int a3, int a4)
- virtual ~playerdata ()

6.14.2 *

Public Attributes

- int phealth1 =100
- int phealth2 =100
- int a
- int b
- std::string t1 ="a"
- bool t2 =true
- GameDataRef ∗ data
- Game ∗ temp

6.14.3 Detailed Description

Struct for testing player data.

6.14.4 Constructor & Destructor Documentation

6.14.4.1 playerdata()

```
stickman::playerdata::playerdata (
             int a1,
             int a2,
             int a3,
             int a4 )  [inline]
```

6.14.4.2 ∼playerdata()

```
virtual stickman::playerdata::∼playerdata ( )  [inline], [virtual]
```

6.14.5 Member Data Documentation

6.14.5.1 a

```
int stickman::playerdata::a
```

6.14.5.2 b

```
int stickman::playerdata::b
```

6.14.5.3 data

```
GameDataRef* stickman::playerdata::data
```

6.14.5.4 phealth1

```
int stickman::playerdata::phealth1 =100
```

6.14.5.5 phealth2

```
int stickman::playerdata::phealth2 =100
```

6.14.5.6 t1

```
std::string stickman::playerdata::t1 ="a"
```

6.14.5.7 t2

```
bool stickman::playerdata::t2 =true
```

6.14.5.8 temp

```
Game* stickman::playerdata::temp
```

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

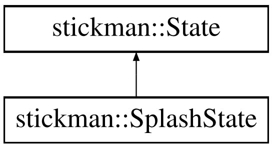
- tests.hpp

6.15 stickman::SplashState Class Reference

Class for splash state.

```
#include <SplashState.hpp>
```

Inheritance diagram for stickman::SplashState:



6.15.1 *

Public Member Functions

- SplashState (GameDataRef data)
  Constructs the object.
- void Init ()
  Virtual function init that may be overloaded which runs at the start of the state.
- void HandleInput ()
  Virtual function HandleInput that may be overloaded which may be used to handle some input.
- void Update (float dt)
  Virtual function Update which may be overloaded which may be used to update game logic.
- void Draw (float dt)
  Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

6.15.2 Detailed Description

Class for splash state.

6.15.3 Constructor & Destructor Documentation

6.15.3.1 SplashState()

```
stickman::SplashState::SplashState (
            GameDataRef data )
```

Constructs the object.

Parameters

in	*data*	The data which contains information about the game

6.15.4 Member Function Documentation

6.15.4.1 Draw()

```
void stickman::SplashState::Draw (
             float dt )  [virtual]
```

Virtual function draw which may be overloaded which may be used to draw something on screen on each iteration.

Parameters

in	*dt*	The difference in frames to syncronise with framerate

Implements stickman::State.

6.15.4.2 HandleInput()

```
void stickman::SplashState::HandleInput ( )  [virtual]
```

Virtual function HandleInput that may be overloaded which may be used to handle some input.

Implements stickman::State.

6.15.4.3 Init()

```
void stickman::SplashState::Init ( )  [virtual]
```

Virtual function init that may be overloaded which runs at the start of the state.

Implements stickman::State.

6.15.4.4 Update()

```
void stickman::SplashState::Update (
             float dt )  [virtual]
```

Virtual function Update which may be overloaded which may be used to update game logic.

Parameters

in	*dt*	The difference in frames to syncronise with framerate

Implements stickman::State.

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

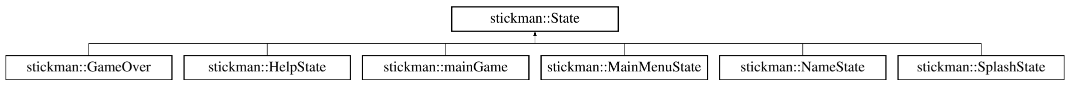
- SplashState.hpp
- SplashState.cpp

6.16 stickman::State Class Reference

Class for state which has functions which can be overloaded so that a particular state of the game can run using these functions.

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

Inheritance diagram for stickman::State:



6.16.1 *

Public Member Functions

- virtual void Init ()=0

  A virtual fucntion which runs at the start when a state is loaded.
- virtual void HandleInput ()=0

  A virtual function which may be used to handle input during each iteration.
- virtual void Update (float dt)=0

  A virtual funciton which may be used to update game logic.
- virtual void Draw (float dt)=0

  A virtual function which may be used to draw objects.
- virtual void Pause ()

  A function which may be used to pause a state.
- virtual void Resume ()

  A function which may be used to resume a state.

6.16.2 Detailed Description

Class for state which has functions which can be overloaded so that a particular state of the game can run using these functions.

6.16.3 Member Function Documentation

6.16.3.1 Draw()

```
virtual void stickman::State::Draw (
            float dt )  [pure virtual]
```

A virtual function which may be used to draw objects.

Parameters

in	*dt*	The difference in frames to syncronise with framerate

Implemented in stickman::MainMenuState, stickman::NameState, stickman::GameOver, stickman::mainGame, stickman::HelpState, and stickman::SplashState.

6.16.3.2 HandleInput()

```
virtual void stickman::State::HandleInput ( )  [pure virtual]
```

A virtual function which may be used to handle input during each iteration.

Implemented in stickman::NameState, stickman::MainMenuState, stickman::GameOver, stickman::mainGame, stickman::HelpState, and stickman::SplashState.

6.16.3.3 Init()

```
virtual void stickman::State::Init ( )  [pure virtual]
```

A virtual fucntion which runs at the start when a state is loaded.

Implemented in stickman::NameState, stickman::MainMenuState, stickman::GameOver, stickman::mainGame, stickman::HelpState, and stickman::SplashState.

6.16.3.4 Pause()

```
virtual void stickman::State::Pause ( )  [inline], [virtual]
```

A function which may be used to pause a state.

6.16.3.5 Resume()

```
virtual void stickman::State::Resume ( )  [inline], [virtual]
```

A function which may be used to resume a state.

6.16.3.6 Update()

```
virtual void stickman::State::Update (
             float dt )  [pure virtual]
```

A virtual funciton which may be used to update game logic.

Parameters

in	*dt*	The difference in frames to syncronise with framerate

Implemented in stickman::NameState, stickman::MainMenuState, stickman::GameOver, stickman::mainGame, stickman::HelpState, and stickman::SplashState.

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

- State.hpp

6.17 stickman::StateMachine Class Reference

Class which is responsible for running a state when it gets loaded.

```
#include <StateMachine.hpp>
```

6.17.1 *

Public Member Functions

- StateMachine ()

  Constructs the object.
- ~StateMachine ()

  Destroys the object.
- void AddState (StateRef newState, bool isReplacing=true)

  Marks a state for adding.
- void RemoveState ()

  Marks a state for removal.
- void ProcessStateChanges ()

  This is the function which replaces the states and adds new stats while deleting previous ones.
- StateRef & GetActiveState ()

  Gets the active state which is running.

6.17.2 Detailed Description

Class which is responsible for running a state when it gets loaded.

6.17.3 Constructor & Destructor Documentation

6.17.3.1 StateMachine()

```
stickman::StateMachine::StateMachine ( )  [inline]
```

Constructs the object.

6.17.3.2 ∼StateMachine()

```
stickman::StateMachine::∼StateMachine ( )  [inline]
```

Destroys the object.

6.17.4 Member Function Documentation

6.17.4.1 AddState()

```
void stickman::StateMachine::AddState (
            StateRef newState,
            bool isReplacing = true )
```

Marks a state for adding.

Parameters

in	*newState*	The new state which will be added
in	*isReplacing*	Indicates if replacing the old state or just pausing it

6.17.4.2 GetActiveState()

```
StateRef & stickman::StateMachine::GetActiveState ( )
```

Gets the active state which is running.

Returns

The active state.

6.17.4.3 ProcessStateChanges()

```
void stickman::StateMachine::ProcessStateChanges ( )
```

This is the function which replaces the states and adds new stats while deleting previous ones.

It may also pause or resume states

6.17.4.4 RemoveState()

```
void stickman::StateMachine::RemoveState ( )
```

Marks a state for removal.

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

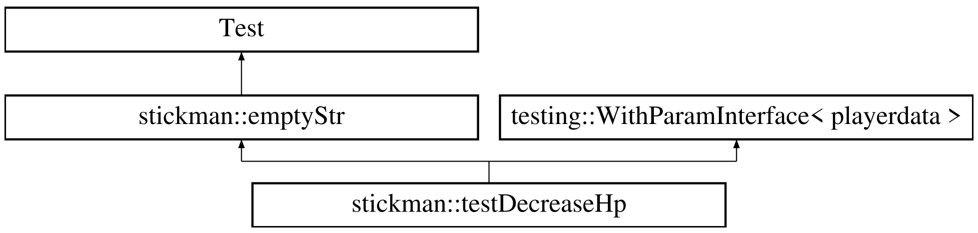
- StateMachine.hpp
- StateMachine.cpp

6.18 stickman::testDecreaseHp Struct Reference

This will be passed to the test as we want an interface to the previous struct.

```
#include <tests.hpp>
```

Inheritance diagram for stickman::testDecreaseHp:



6.18.1 Detailed Description

This will be passed to the test as we want an interface to the previous struct.

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

- tests.hpp

Chapter 7

File Documentation

7.1 AssetManager.cpp File Reference

```
#include <SFML/Graphics.hpp>
#include "AssetManager.hpp"
```

7.1.1 *

Namespaces

- stickman

7.2 AssetManager.hpp File Reference

```
#include <map>
#include <SFML/Graphics.hpp>
```

7.2.1 *

Classes

- class stickman::AssetManager

  Class for asset manager.This class loads a texture and creates a map between textures and strings so that we don't have to load the same texture and sprite again and again.

7.2.2 *

Namespaces

- stickman

7.3 DEFINITIONS.hpp File Reference

7.3.1 *

Macros

- #define SCREEN_WIDTH 1366
- #define SCREEN_HEIGHT 768
- #define SPLASH_STATE_SHOW_TIME 10.0
- #define SPLASH_SCENE_BACKGROUND_FILEPATH "res/fight.png"

  The assets for the splash screen state.
- #define SPLASH_SCENE_LOGO_FILEPATH "res/logo.png"
- #define SPLASH_SCENE_Press_FILEPATH "res/press1.png"
- #define Restart_FILEPATH "res/restart.png"
- #define SPLASH_SCENE_CREATE_FILEPATH "res/CREATED.png"
- #define SPLASH_SCENE_NAME1_FILEPATH "res/shivashish.png"
- #define SPLASH_SCENE_NAME2_FILEPATH "res/ajinkya.png"
- #define SPLASH_SCENE_NAME3_FILEPATH "res/tungadri.png"
- #define SPLASH_SCENE_NAME4_FILEPATH "res/niraj.png"
- #define SPLASH_SCENE_BALOON_FILEPATH "res/baloon.png"

  Assets for the main menu state.
- #define SPLASH_SCENE_BALOON1_FILEPATH "res/baloon1.png"
- #define ENTER_BUTTON_1_FILEPATH "res/enter.png"
- #define BACK_BUTTON "res/back_button.png"
- #define HOST_BUTTON "res/host.png"
- #define JOIN_BUTTON "res/join.png"
- #define HELP_BUTTON "res/help.png"

7.3.2 Macro Definition Documentation

7.3.2.1 BACK_BUTTON

```
#define BACK_BUTTON "res/back_button.png"
```

7.3.2.2 ENTER_BUTTON_1_FILEPATH

```
#define ENTER_BUTTON_1_FILEPATH "res/enter.png"
```

7.3.2.3 HELP_BUTTON

```
#define HELP_BUTTON "res/help.png"
```

7.3.2.4 HOST_BUTTON

```
#define HOST_BUTTON "res/host.png"
```

7.3.2.5 JOIN_BUTTON

```
#define JOIN_BUTTON "res/join.png"
```

7.3.2.6 Restart_FILEPATH

```
#define Restart_FILEPATH "res/restart.png"
```

7.3.2.7 SCREEN_HEIGHT

```
#define SCREEN_HEIGHT 768
```

7.3.2.8 SCREEN_WIDTH

```
#define SCREEN_WIDTH 1366
```

7.3.2.9 SPLASH_SCENE_BACKGROUND_FILEPATH

```
#define SPLASH_SCENE_BACKGROUND_FILEPATH "res/fight.png"
```

The assets for the splash screen state.

7.3.2.10 SPLASH_SCENE_BALOON1_FILEPATH

```
#define SPLASH_SCENE_BALOON1_FILEPATH "res/baloon1.png"
```

7.3.2.11 SPLASH_SCENE_BALOON_FILEPATH

```
#define SPLASH_SCENE_BALOON_FILEPATH "res/baloon.png"
```

Assets for the main menu state.

7.3.2.12 SPLASH_SCENE_CREATE_FILEPATH

```
#define SPLASH_SCENE_CREATE_FILEPATH "res/CREATED.png"
```

7.3.2.13 SPLASH_SCENE_LOGO_FILEPATH

```
#define SPLASH_SCENE_LOGO_FILEPATH "res/logo.png"
```

7.3.2.14 SPLASH_SCENE_NAME1_FILEPATH

```
#define SPLASH_SCENE_NAME1_FILEPATH "res/shivashish.png"
```

7.3.2.15 SPLASH_SCENE_NAME2_FILEPATH

```
#define SPLASH_SCENE_NAME2_FILEPATH "res/ajinkya.png"
```

7.3.2.16 SPLASH_SCENE_NAME3_FILEPATH

```
#define SPLASH_SCENE_NAME3_FILEPATH "res/tungadri.png"
```

7.3.2.17 SPLASH_SCENE_NAME4_FILEPATH

```
#define SPLASH_SCENE_NAME4_FILEPATH "res/niraj.png"
```

7.3.2.18 SPLASH_SCENE_Press_FILEPATH

```
#define SPLASH_SCENE_Press_FILEPATH "res/press1.png"
```

7.3.2.19 SPLASH_STATE_SHOW_TIME

```
#define SPLASH_STATE_SHOW_TIME 10.0
```

7.4 Game.cpp File Reference

```
#include "Game.hpp"
#include "SplashState.hpp"
```

7.4.1 *

Namespaces

- stickman

7.5 game.cpp File Reference

```
#include "player.h"
#include "SFML/Graphics.hpp"
#include "SFML/Network.hpp"
#include "game.h"
#include "Box2D/Box2D.h"
#include "myListener.h"
#include <iostream>
```

7.5.1 *

Namespaces

- stickman

7.5.2 *

Macros

- #define DEGTORAD 0.0174532925199432957f

7.5.3 *

Variables

- const float SCALE = 30.f
- float temp1 = ((75/2)/sqrt(2))
- float temp2 = (60/2)

7.5.4 Macro Definition Documentation

7.5.4.1 DEGTORAD

```
#define DEGTORAD 0.0174532925199432957f
```

7.5.5 Variable Documentation

7.5.5.1 SCALE

```
const float SCALE = 30.f
```

7.5.5.2 temp1

```
float temp1 = ((75/2)/sqrt(2))
```

7.5.5.3 temp2

```
float temp2 = (60/2)
```

7.6 game.h File Reference

```
#include "SFML/Graphics.hpp"
#include "Box2D/Box2D.h"
#include "player.h"
#include "myListener.h"
#include "SFML/Network.hpp"
#include "SFML/Audio.hpp"
#include <queue>
#include <time.h>
#include <sys/time.h>
#include <thread>
#include <mutex>
#include "State.hpp"
#include "Game.hpp"
#include <utility>
#include <string>
```

7.6.1 *

Classes

- class stickman::Game

  Contains all basic entities required in game like window, players object, TcpListener, Send and receive sockets, Box2D world, walls,ground, sprites of all bodies to be displayed in window and functions to check collision, decrease health points, sending and receving packets from client to server and vice versa, worker threads which checks for collision and gem thread used to generate gem.

7.6.2 *

Namespaces

- stickman

7.7 Game.hpp File Reference

```
#include <memory>
#include <string>
#include "SFML/Graphics.hpp"
#include "StateMachine.hpp"
#include "AssetManager.hpp"
#include "InputManager.hpp"
```

7.7.1 *

Classes

- struct stickman::GameData

  This contains the objects required by the game as the whole like the state machine which switches states and different managers to make loading different things easier.
- class stickman::Game2

  Class for game which initializes the different properties related to the game like resolution and so on.

7.7.2 *

Namespaces

- stickman

7.7.3 *

Typedefs

- typedef std::shared_ptr< GameData > stickman::GameDataRef

  Creating container for raw pointers for the struct game data.

7.8 GameOver.cpp File Reference

```
#include <sstream>
#include "GameOver.h"
#include "DEFINITIONS.hpp"
#include "name.h"
#include <iostream>
```

7.8.1 *

Namespaces

- stickman

7.9 GameOver.h File Reference

```
#include <SFML/Graphics.hpp>
#include "State.hpp"
#include "mainGame.hpp"
#include "Game.hpp"
#include <string>
```

7.9.1 *

Classes

- class stickman::GameOver

  Class for game over state.

7.9.2 *

Namespaces

- stickman

7.10 HelpState.cpp File Reference

```
#include <sstream>
#include "HelpState.hpp"
#include "name.h"
#include "DEFINITIONS.hpp"
#include <iostream>
#include <string>
```

7.10.1 *

Namespaces

- stickman

7.11 HelpState.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "State.hpp"
#include "name.h"
#include "Game.hpp"
#include <string>
```

7.11.1 *

Classes

- class stickman::HelpState

  Class for help state.

7.11.2 *

Namespaces

- stickman

7.12 InputManager.cpp File Reference

```
#include "InputManager.hpp"
```

7.12.1 *

Namespaces

- stickman

7.13 InputManager.hpp File Reference

```
#include "SFML/Graphics.hpp"
```

7.13.1 *

Classes

- class stickman::InputManager

  Class for input manager.

7.13.2 *

Namespaces

- stickman

7.14 mainGame.cpp File Reference

```
#include "mainGame.hpp"
```

7.14.1 *

Namespaces

- stickman

7.15 mainGame.hpp File Reference

```
#include "SFML/Graphics.hpp"
#include "Box2D/Box2D.h"
#include "player.h"
#include "game.h"
#include <sstream>
#include "State.hpp"
#include "Game.hpp"
#include "GameOver.h"
#include <bits/stdc++.h>
```

7.15.1 *

Classes

- struct stickman::mainGame

    Class for main game.

7.15.2 *

Namespaces

- stickman

7.15.3 *

Macros

- #define DEGTORAD 0.0174532925199432957f
- #define RADTODEG 57.295779513082320876f

7.15.4 *

Variables

- const float SCALE = 30.f

7.15.5 Macro Definition Documentation

7.15.5.1 DEGTORAD

```
#define DEGTORAD 0.0174532925199432957f
```

7.15.5.2 RADTODEG

```
#define RADTODEG 57.295779513082320876f
```

7.15.6 Variable Documentation

7.15.6.1 SCALE

```
const float SCALE = 30.f
```

7.16 MainMenuState.cpp File Reference

```
#include <sstream>
#include "MainMenuState.hpp"
#include "DEFINITIONS.hpp"
#include <iostream>
```

7.16.1 *

Namespaces

- stickman

7.17 MainMenuState.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "State.hpp"
#include "mainGame.hpp"
#include "Game.hpp"
#include <string>
#include "name.h"
```

7.17.1 *

Classes

- class stickman::MainMenuState
    Class for main menu state.

7.17.2 *

Namespaces

- stickman

7.18 myListener.cpp File Reference

```
#include "myListener.h"
#include "Box2D/Box2D.h"
```

7.18.1 *

Namespaces

- stickman

7.19 myListener.h File Reference

```
#include "Box2D/Box2D.h"
#include <queue>
#include <utility>
#include <iostream>
```

7.19.1 *

Classes

- class stickman::myListener

  Listens to collision between any two objects in Box2D world.

7.19.2 *

Namespaces

- stickman

7.20 name.cpp File Reference

```
#include <sstream>
#include "name.h"
#include "DEFINITIONS.hpp"
#include "MainMenuState.hpp"
#include <iostream>
```

7.20.1 *

Namespaces

- stickman

7.21 name.h File Reference

```
#include "State.hpp"
#include "mainGame.hpp"
#include <HelpState.hpp>
#include "Game.hpp"
#include <string>
#include "SFML/Graphics.hpp"
```

7.21.1 *

Classes

- class stickman::NameState

  Class for name state which takes the name of player and gives the option of chosing whether to host a server/ join a server.

7.21.2 *

Namespaces

- stickman

7.22 player.cpp File Reference

```
#include "player.h"
#include "SFML/Graphics.hpp"
#include "Box2D/Box2D.h"
```

7.22.1 *

Namespaces

- stickman

7.22.2 *

Macros

- #define DEGTORAD 0.0174532925199432957f

7.22.3 *

Variables

- const float SCALE = 30.f

7.22.4 Macro Definition Documentation

7.22.4.1 DEGTORAD

```
#define DEGTORAD 0.0174532925199432957f
```

7.22.5 Variable Documentation

7.22.5.1 SCALE

```
const float SCALE = 30.f
```

7.23 player.h File Reference

```
#include "SFML/Graphics.hpp"
#include "Box2D/Box2D.h"
#include "string"
#include <iostream>
```

7.23.1 *

Classes

- class stickman::Player

  Contains all the information about the player.

7.23.2 *

Namespaces

- stickman

7.24 SplashState.cpp File Reference

```
#include <sstream>
#include "SplashState.hpp"
#include "DEFINITIONS.hpp"
#include "MainMenuState.hpp"
#include "name.h"
#include <iostream>
#include <SFML/Graphics.hpp>
```

7.24.1 *

Namespaces

- stickman

7.24.2 *

Variables

- int a =255

7.24.3 Variable Documentation

7.24.3.1 a

```
int a =255
```

7.25 SplashState.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "State.hpp"
#include "Game.hpp"
#include "name.h"
```

7.25.1 *

Classes

- class stickman::SplashState

    Class for splash state.

7.25.2 *

Namespaces

- stickman

7.26 State.hpp File Reference

7.26.1 *

Classes

- class stickman::State

    Class for state which has functions which can be overloaded so that a particular state of the game can run using these functions.

7.26.2 *

Namespaces

- stickman

7.27 StateMachine.cpp File Reference

```
#include "StateMachine.hpp"
```

7.27.1 *

Namespaces

- stickman

7.28 StateMachine.hpp File Reference

```
#include <memory>
#include <stack>
#include "State.hpp"
```

7.28.1 *

Classes

- class stickman::StateMachine

  Class which is responsible for running a state when it gets loaded.

7.28.2 *

Namespaces

- stickman

7.28.3 *

Typedefs

- typedef std::unique_ptr< State > stickman::StateRef

  Creates a unique pointer for StateRef so that it gets automatically destroyed.

7.29 tests.hpp File Reference

```
#include "game.h"
#include <gtest/gtest.h>
#include "SFML/Graphics.hpp"
#include "Box2D/Box2D.h"
#include "player.h"
#include "myListener.h"
#include "SFML/Network.hpp"
#include <queue>
#include <time.h>
#include <sys/time.h>
#include <thread>
#include <mutex>
#include "State.hpp"
#include "Game.hpp"
#include <utility>
#include <bits/stdc++.h>
```

7.29.1 *

Classes

- struct stickman::mainGame

    Class for main game.
- struct stickman::emptyStr

    An empty struct to derive from.
- struct stickman::playerdata

    Struct for testing player data.
- struct stickman::testDecreaseHp

    This will be passed to the test as we want an interface to the previous struct.

7.29.2 *

Namespaces

- stickman

7.29.3 *

Macros

- #define temp_h

7.29.4 *

Functions

- stickman::TEST_F (mainGame, initializeData)

  Runs the test initializeData.
- stickman::TEST_F (mainGame, generateGem)

  test for generate gem function
- stickman::TEST_F (mainGame, checkDistance)

  Test for the distance function.
- stickman::TEST_P (testDecreaseHp, decreaseHp)

  Performs a test with multiple inputs to check different test cases.
- stickman::INSTANTIATE_TEST_CASE_P (Default, testDecreaseHp, testing::Values(playerdata{100, 100, 1, 5}, playerdata{100, 100, 5, 1}, playerdata{100, 100, 4, 6}, playerdata{100, 100, 6, 4}, playerdata{100, 100, 6, 3}, playerdata{100, 100, 3, 6}, playerdata{100, 100, 4, 5}, playerdata{100, 100, 5, 4}, playerdata{100, 100, 3, 5}, playerdata{100, 100, 5, 3}, playerdata{100, 100, 2, 8}, playerdata{100, 100, 8, 2}, playerdata{100, 100, 7, 2}, playerdata{100, 100, 2, 7}, playerdata{100, 100, 1, 7}, playerdata{100, 100, 7, 1}, playerdata{80, 100, 4, 6}, playerdata{100, 80, 6, 4}, playerdata{100, 60, 4, 6}, playerdata{60, 100, 6, 4}, playerdata{100, 40, 4, 6}, playerdata{40, 100, 6, 4}, playerdata{100, 20, 4, 6}, playerdata{20, 100, 6, 4}, playerdata{100, 10, 4, 6}, playerdata{10, 100, 6, 4}, playerdata{80, 80, 4, 6}, playerdata{80, 80, 6, 4}, playerdata{80, 60, 4, 6}, playerdata{60, 80, 6, 4}, playerdata{80, 40, 4, 6}, playerdata{40, 80, 6, 4}, playerdata{80, 20, 4, 6}, playerdata{20, 80, 6, 4}, playerdata{80, 10, 4, 6}, playerdata{10, 80, 6, 4}, playerdata{60, 80, 4, 6}, playerdata{80, 60, 6, 4}, playerdata{60, 60, 4, 6}, playerdata{60, 60, 6, 4}, playerdata{60, 40, 4, 6}, playerdata{40, 60, 6, 4}, playerdata{60, 20, 4, 6}, playerdata{20, 60, 6, 4}, playerdata{60, 10, 4, 6}, playerdata{10, 60, 6, 4}, playerdata{40, 80, 4, 6}, playerdata{80, 40, 6, 4}, playerdata{40, 60, 4, 6}, playerdata{60, 40, 6, 4}))

  Passes the test cases for the test.

7.29.5 Macro Definition Documentation

7.29.5.1 temp_h

```
#define temp_h
```