

COMP-345

v0.0.0

Generated by Doxygen 1.13.2



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Map::Cell Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Data Documentation	7
4.1.2.1 flowDirectionX	7
4.1.2.2 flowDirectionY	7
4.1.2.3 flowDistance	8
4.1.2.4 isSpawner	8
4.1.2.5 isTarget	8
4.1.2.6 isWall	8
4.1.2.7 x	8
4.1.2.8 y	8
4.2 ExitState Class Reference	9
4.2.1 Constructor & Destructor Documentation	10
4.2.1.1 ExitState()	10
4.2.2 Member Function Documentation	10
4.2.2.1 enter()	10
4.2.2.2 exit()	10
4.2.2.3 get()	11
4.2.2.4 handleEvent()	11
4.2.2.5 render()	11
4.2.2.6 update()	11
4.2.3 Member Data Documentation	12
4.2.3.1 sExitState	12
4.3 GameState Class Reference	12
4.3.1 Constructor & Destructor Documentation	12
4.3.1.1 ~GameState()	12
4.3.2 Member Function Documentation	12
4.3.2.1 enter()	12
4.3.2.2 exit()	13
4.3.2.3 handleEvent()	13
4.3.2.4 render()	13
4.3.2.5 update()	13
4.4 Global Class Reference	13

4.4.1 Member Data Documentation	13
4.4.1.1 kScreenHeight	13
4.4.1.2 kScreenWidth	14
4.5 IntroState Class Reference	14
4.5.1 Constructor & Destructor Documentation	15
4.5.1.1 IntroState()	15
4.5.2 Member Function Documentation	15
4.5.2.1 enter()	15
4.5.2.2 exit()	16
4.5.2.3 get()	16
4.5.2.4 handleEvent()	16
4.5.2.5 render()	17
4.5.2.6 update()	17
4.5.3 Member Data Documentation	17
4.5.3.1 mBackgroundTexture	17
4.5.3.2 mMessageTexture	17
4.5.3.3 sIntroState	17
4.6 LButton Class Reference	17
4.6.1 Member Enumeration Documentation	18
4.6.1.1 eButtonSprite	18
4.6.2 Constructor & Destructor Documentation	18
4.6.2.1 LButton()	18
4.6.3 Member Function Documentation	19
4.6.3.1 handleEvent()	19
4.6.3.2 isClicked()	19
4.6.3.3 render()	19
4.6.3.4 setPosition()	19
4.6.3.5 setText()	19
4.6.4 Member Data Documentation	19
4.6.4.1 gButtonSpriteTexture	19
4.6.4.2 kButtonHeight	19
4.6.4.3 kButtonWidth	19
4.6.4.4 mCurrentSprite	20
4.6.4.5 mPosition	20
4.7 LTexture Class Reference	20
4.7.1 Constructor & Destructor Documentation	20
4.7.1.1 LTexture()	20
4.7.1.2 ~LTexture()	21
4.7.2 Member Function Documentation	21
4.7.2.1 destroy()	21
4.7.2.2 getHeight()	21
4.7.2.3 getWidth()	21

4.7.2.4 loadFromFile()	22
4.7.2.5 loadFromRenderedText()	22
4.7.2.6 render()	23
4.7.2.7 setAlpha()	23
4.7.2.8 setBlending()	23
4.7.2.9 setColor()	23
4.7.3 Member Data Documentation	23
4.7.3.1 kOriginalSize	23
4.7.3.2 mHeight	24
4.7.3.3 mTexture	24
4.7.3.4 mWidth	24
4.8 LTimer Class Reference	24
4.8.1 Constructor & Destructor Documentation	24
4.8.1.1 LTimer()	24
4.8.2 Member Function Documentation	25
4.8.2.1 getTicksNS()	25
4.8.2.2 isPaused()	25
4.8.2.3 isStarted()	25
4.8.2.4 pause()	25
4.8.2.5 start()	25
4.8.2.6 stop()	26
4.8.2.7 unpause()	26
4.8.3 Member Data Documentation	26
4.8.3.1 mPaused	26
4.8.3.2 mPausedTicks	26
4.8.3.3 mStarted	26
4.8.3.4 mStartTicks	26
4.9 MainGameState Class Reference	27
4.9.1 Constructor & Destructor Documentation	28
4.9.1.1 MainGameState()	28
4.9.2 Member Function Documentation	28
4.9.2.1 enter()	28
4.9.2.2 exit()	28
4.9.2.3 get()	29
4.9.2.4 handleEvent()	29
4.9.2.5 render()	29
4.9.2.6 update()	29
4.9.3 Member Data Documentation	30
4.9.3.1 mBackgroundTexture	30
4.9.3.2 mMessageTexture	30
4.9.3.3 sMainGameState	30
4.10 Map Class Reference	30

4.10.1 Constructor & Destructor Documentation	31
4.10.1.1 Map()	31
4.10.2 Member Function Documentation	32
4.10.2.1 calculateDistances()	32
4.10.2.2 calculateFlowDirections()	33
4.10.2.3 calculateFlowField()	33
4.10.2.4 draw()	34
4.10.2.5 drawCell()	34
4.10.2.6 getFlowNormal()	35
4.10.2.7 getTargetPos()	36
4.10.2.8 isCellWall()	36
4.10.2.9 isInbounds()	36
4.10.2.10 isSpawner()	37
4.10.2.11 isTarget()	38
4.10.2.12 isValidPath()	39
4.10.2.13 setCellWall()	39
4.10.2.14 setSpawner()	40
4.10.2.15 setTarget()	40
4.10.3 Member Data Documentation	41
4.10.3.1 cellCountX	41
4.10.3.2 cellCountY	41
4.10.3.3 cells	41
4.10.3.4 flowDistanceMax	42
4.10.3.5 PIXELS_PER_CELL	42
4.10.3.6 textureCellArrowDown	42
4.10.3.7 textureCellArrowLeft	42
4.10.3.8 textureCellArrowRight	42
4.10.3.9 textureCellArrowUp	42
4.10.3.10 textureCellEmpty	42
4.10.3.11 textureCellSpawner	42
4.10.3.12 textureCellTarget	43
4.10.3.13 textureCellWall	43
4.11 Part1State Class Reference	43
4.11.1 Constructor & Destructor Documentation	45
4.11.1.1 Part1State()	45
4.11.2 Member Function Documentation	45
4.11.2.1 enter()	45
4.11.2.2 exit()	45
4.11.2.3 get()	46
4.11.2.4 handleEvent()	46
4.11.2.5 render()	47
4.11.2.6 update()	47

4.11.3 Member Data Documentation	47
4.11.3.1 keyDownStatus	47
4.11.3.2 map	47
4.11.3.3 mBackgroundTexture	47
4.11.3.4 mMessageTexture	48
4.11.3.5 mouseDownStatus	48
4.11.3.6 sPart1State	48
4.12 Part2State Class Reference	48
4.12.1 Constructor & Destructor Documentation	49
4.12.1.1 Part2State()	49
4.12.2 Member Function Documentation	50
4.12.2.1 enter()	50
4.12.2.2 exit()	50
4.12.2.3 get()	50
4.12.2.4 handleEvent()	50
4.12.2.5 render()	51
4.12.2.6 update()	51
4.12.3 Member Data Documentation	51
4.12.3.1 mBackgroundTexture	51
4.12.3.2 mMessageTexture	51
4.12.3.3 sPart2State	51
4.13 Part3State Class Reference	51
4.13.1 Constructor & Destructor Documentation	53
4.13.1.1 Part3State()	53
4.13.2 Member Function Documentation	53
4.13.2.1 enter()	53
4.13.2.2 exit()	53
4.13.2.3 get()	53
4.13.2.4 handleEvent()	54
4.13.2.5 render()	54
4.13.2.6 update()	54
4.13.3 Member Data Documentation	54
4.13.3.1 mBackgroundTexture	54
4.13.3.2 mMessageTexture	54
4.13.3.3 sPart3State	54
4.14 TextureLoader Class Reference	54
4.14.1 Member Function Documentation	55
4.14.1.1 deallocateTextures()	55
4.14.1.2 loadTexture()	56
4.14.2 Member Data Documentation	57
4.14.2.1 loadedTextures	57
4.14.2.2 TEXTURE_PATH	57

4.15 TitleState Class Reference . . . . .	57
4.15.1 Constructor & Destructor Documentation . . . . .	58
4.15.1.1 TitleState() . . . . .	58
4.15.2 Member Function Documentation . . . . .	59
4.15.2.1 enter() . . . . .	59
4.15.2.2 exit() . . . . .	59
4.15.2.3 get() . . . . .	59
4.15.2.4 handleEvent() . . . . .	60
4.15.2.5 render() . . . . .	60
4.15.2.6 update() . . . . .	60
4.15.3 Member Data Documentation . . . . .	60
4.15.3.1 mBackgroundTexture . . . . .	60
4.15.3.2 mMessageTexture . . . . .	60
4.15.3.3 sTitleState . . . . .	61
4.16 Vector2D Class Reference . . . . .	61
4.16.1 Constructor & Destructor Documentation . . . . .	62
4.16.1.1 Vector2D() [1/4] . . . . .	62
4.16.1.2 Vector2D() [2/4] . . . . .	63
4.16.1.3 Vector2D() [3/4] . . . . .	63
4.16.1.4 Vector2D() [4/4] . . . . .	63
4.16.2 Member Function Documentation . . . . .	64
4.16.2.1 angle() . . . . .	64
4.16.2.2 angleBetween() . . . . .	64
4.16.2.3 cross() . . . . .	65
4.16.2.4 dot() . . . . .	65
4.16.2.5 getNegativeReciprocal() . . . . .	66
4.16.2.6 magnitude() . . . . .	66
4.16.2.7 normalize() . . . . .	67
4.16.2.8 operator*() [1/2] . . . . .	67
4.16.2.9 operator*() [2/2] . . . . .	68
4.16.2.10 operator*=( ) [1/2] . . . . .	68
4.16.2.11 operator*=( ) [2/2] . . . . .	68
4.16.2.12 operator+( ) [1/2] . . . . .	69
4.16.2.13 operator+( ) [2/2] . . . . .	69
4.16.2.14 operator+=( ) [1/2] . . . . .	69
4.16.2.15 operator+=( ) [2/2] . . . . .	70
4.16.2.16 operator-( ) [1/2] . . . . .	70
4.16.2.17 operator-( ) [2/2] . . . . .	70
4.16.2.18 operator-=( ) [1/2] . . . . .	71
4.16.2.19 operator-=( ) [2/2] . . . . .	71
4.16.2.20 operator/( ) [1/2] . . . . .	71
4.16.2.21 operator/( ) [2/2] . . . . .	72



4.16.2.22 operator/=( ) [1/2]	72
4.16.2.23 operator/=( ) [2/2]	72
4.16.3 Member Data Documentation	73
4.16.3.1 x	73
4.16.3.2 y	73
<b>5 File Documentation</b>	<b>75</b>
5.1 src/include/Global.h File Reference	75
5.1.1 Function Documentation	76
5.1.1.1 changeState()	76
5.1.1.2 close()	76
5.1.1.3 init()	77
5.1.1.4 loadMedia()	77
5.1.1.5 setNextState()	77
5.1.2 Variable Documentation	78
5.1.2.1 gFont	78
5.1.2.2 gRenderer	78
5.2 Global.h	78
5.3 src/include/LTimer.h File Reference	79
5.4 LTimer.h	79
5.5 src/include/map/Map.h File Reference	79
5.5.1 Detailed Description	80
5.6 Map.h	81
5.7 src/include/states/ExitState.h File Reference	82
5.8 ExitState.h	83
5.9 src/include/states/GameState.h File Reference	83
5.10 GameState.h	84
5.11 src/include/states/IntroState.h File Reference	84
5.11.1 Macro Definition Documentation	85
5.11.1.1 INTROSTATE_H	85
5.12 IntroState.h	85
5.13 src/include/states/parts/MainGameState.h File Reference	86
5.14 MainGameState.h	87
5.15 src/include/states/parts/Part1State.h File Reference	88
5.15.1 Detailed Description	89
5.16 Part1State.h	89
5.17 src/include/states/parts/Part2State.h File Reference	89
5.18 Part2State.h	90
5.19 src/include/states/parts/Part3State.h File Reference	91
5.20 Part3State.h	92
5.21 src/include/states/TitleState.h File Reference	93
5.22 TitleState.h	94

5.23 src/include/ui/LButton.h File Reference	94
5.24 LButton.h	95
5.25 src/include/ui/LTexture.h File Reference	96
5.26 LTexture.h	96
5.27 src/include/util/TextureLoader.h File Reference	97
5.27.1 Detailed Description	98
5.28 TextureLoader.h	98
5.29 src/include/util/Vector2D.h File Reference	99
5.29.1 Detailed Description	99
5.30 Vector2D.h	100
5.31 src/LTimer.cpp File Reference	100
5.32 src/main.cpp File Reference	101
5.32.1 Function Documentation	101
5.32.1.1 changeState()	101
5.32.1.2 close()	102
5.32.1.3 init()	102
5.32.1.4 loadMedia()	103
5.32.1.5 main()	103
5.32.1.6 setNextState()	104
5.32.2 Variable Documentation	104
5.32.2.1 gCurrentState	104
5.32.2.2 gFont	104
5.32.2.3 gFpsTexture	104
5.32.2.4 gNextState	105
5.32.2.5 gRenderer	105
5.32.2.6 gWindow	105
5.32.2.7 kScreenFps	105
5.33 src/map/Map.cpp File Reference	105
5.33.1 Detailed Description	105
5.34 src/states/ExitState.cpp File Reference	106
5.35 ExitState.cpp	106
5.36 src/states/IntroState.cpp File Reference	107
5.37 IntroState.cpp	108
5.38 src/states/parts/MainGameState.cpp File Reference	109
5.39 src/states/parts/Part1State.cpp File Reference	109
5.39.1 Detailed Description	110
5.40 src/states/parts/Part2State.cpp File Reference	110
5.41 src/states/parts/Part3State.cpp File Reference	111
5.42 src/states/TitleState.cpp File Reference	112
5.42.1 Variable Documentation	112
5.42.1.1 buttons	112
5.42.1.2 kButtonCount	113

---

5.43 TitleState.cpp . . . . .	113
5.44 src/ui/LButton.cpp File Reference . . . . .	115
5.45 src/ui/LTexture.cpp File Reference . . . . .	115
5.46 LTexture.cpp . . . . .	116
5.47 src/util/TextureLoader.cpp File Reference . . . . .	118
5.47.1 Detailed Description . . . . .	118
5.47.2 Function Documentation . . . . .	118
5.47.2.1 getLoadedTextures() . . . . .	118
5.48 src/util/Vector2D.cpp File Reference . . . . .	119
5.48.1 Detailed Description . . . . .	119
<b>Index</b>	<b>121</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Map::Cell . . . . .	7
GameState . . . . .	12
ExitState . . . . .	9
IntroState . . . . .	14
MainGameState . . . . .	27
Part1State . . . . .	43
Part2State . . . . .	48
Part3State . . . . .	51
TitleState . . . . .	57
Global . . . . .	13
LButton . . . . .	17
LTexture . . . . .	20
LTimer . . . . .	24
Map . . . . .	30
TextureLoader . . . . .	54
Vector2D . . . . .	61



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Map::Cell</a>	
Represents a single cell in the map grid . . . . .	7
<a href="#">ExitState</a> . . . . .	9
<a href="#">GameState</a> . . . . .	12
<a href="#">Global</a> . . . . .	13
<a href="#">IntroState</a> . . . . .	14
<a href="#">LButton</a> . . . . .	17
<a href="#">LTexture</a> . . . . .	20
<a href="#">LTimer</a> . . . . .	24
<a href="#">MainGameState</a> . . . . .	27
<a href="#">Map</a> . . . . .	30
<a href="#">Part1State</a> . . . . .	43
<a href="#">Part2State</a> . . . . .	48
<a href="#">Part3State</a> . . . . .	51
<a href="#">TextureLoader</a> . . . . .	54
<a href="#">TitleState</a> . . . . .	57
<a href="#">Vector2D</a> . . . . .	61





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

src/LTimer.cpp	100
src/main.cpp	101
src/include/Global.h	75
src/include/LTimer.h	79
src/include/map/Map.h	
Responsible for generating the tower defense map	79
src/include/states/ExitState.h	82
src/include/states/GameState.h	83
src/include/states/IntroState.h	84
src/include/states/TitleState.h	93
src/include/states/parts/MainGameState.h	86
src/include/states/parts/Part1State.h	
The header file of part 1's (Map) driver	88
src/include/states/parts/Part2State.h	89
src/include/states/parts/Part3State.h	91
src/include/ui/LButton.h	94
src/include/ui/LTexture.h	96
src/include/util/TextureLoader.h	
Load and store textures	97
src/include/util/Vector2D.h	
Representation of a 2D vector	99
src/map/Map.cpp	
Implementation of the Map class	105
src/states/ExitState.cpp	106
src/states/IntroState.cpp	107
src/states/TitleState.cpp	112
src/states/parts/MainGameState.cpp	109
src/states/parts/Part1State.cpp	
The drive file for part 1 of assignment 1 (Map)	109
src/states/parts/Part2State.cpp	110
src/states/parts/Part3State.cpp	111
src/ui/LButton.cpp	115
src/ui/LTexture.cpp	115
src/util/TextureLoader.cpp	
Implementation of the Texture Loader Class	118
src/util/Vector2D.cpp	
Implementation of a the Vector2D class	119



## Chapter 4

# Class Documentation

### 4.1 Map::Cell Struct Reference

Represents a single cell in the map grid.

#### Public Attributes

- int `x` = 0
- int `y` = 0
- bool `isWall` = false
- bool `isSpawner` = false
- bool `isTarget` = false
- int `flowDirectionX` = 0
- int `flowDirectionY` = 0
- unsigned char `flowDistance` = `flowDistanceMax`

#### 4.1.1 Detailed Description

Represents a single cell in the map grid.

Contains position, type, and flow field information for pathfinding

#### 4.1.2 Member Data Documentation

##### 4.1.2.1 flowDirectionX

```
int Map::Cell::flowDirectionX = 0
```

X component of flow direction

##### 4.1.2.2 flowDirectionY

```
int Map::Cell::flowDirectionY = 0
```

Y component of flow direction

#### 4.1.2.3 flowDistance

```
unsigned char Map::Cell::flowDistance = flowDistanceMax
```

Distance to target in flow field

#### 4.1.2.4 isSpawner

```
bool Map::Cell::isSpawner = false
```

Whether this cell is an enemy spawner

#### 4.1.2.5 isTarget

```
bool Map::Cell::isTarget = false
```

Whether this cell is a target

#### 4.1.2.6 isWall

```
bool Map::Cell::isWall = false
```

Whether this cell is a wall

#### 4.1.2.7 x

```
int Map::Cell::x = 0
```

X coordinate in the grid

#### 4.1.2.8 y

```
int Map::Cell::y = 0
```

Y coordinate in the grid

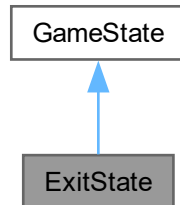
The documentation for this struct was generated from the following file:

- [src/include/map/Map.h](#)

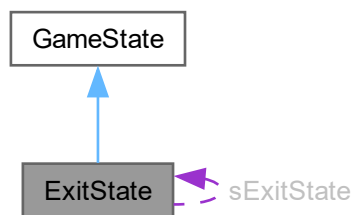
## 4.2 ExitState Class Reference

```
#include <ExitState.h>
```

Inheritance diagram for ExitState:



Collaboration diagram for ExitState:



### Public Member Functions

- bool [enter](#) () override
- bool [exit](#) () override
- void [handleEvent](#) (SDL\_Event &e) override
- void [update](#) () override
- void [render](#) () override

### Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

### Static Public Member Functions

- static [ExitState](#) \* [get](#) ()

## Private Member Functions

- [ExitState](#) ()

## Static Private Attributes

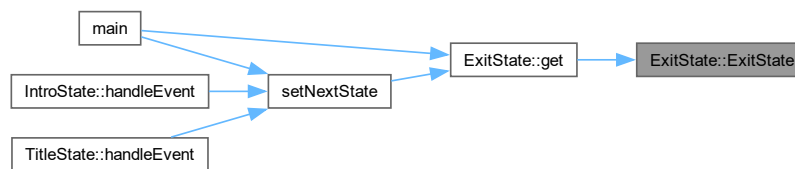
- static [ExitState](#) `sExitState`

## 4.2.1 Constructor & Destructor Documentation

### 4.2.1.1 ExitState()

```
ExitState::ExitState () [private]
```

Here is the caller graph for this function:



## 4.2.2 Member Function Documentation

### 4.2.2.1 enter()

```
bool ExitState::enter () [override], [virtual]
```

Implements [GameState](#).

### 4.2.2.2 exit()

```
bool ExitState::exit () [override], [virtual]
```

Implements [GameState](#).

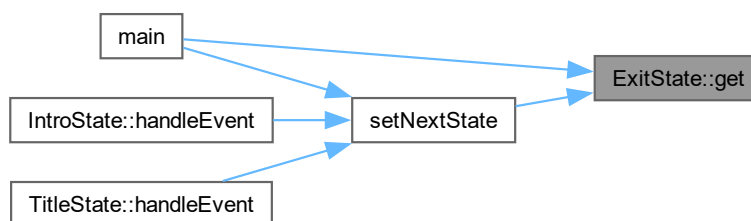
## 4.2.2.3 get()

```
ExitState * ExitState::get () [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.2.2.4 handleEvent()

```
void ExitState::handleEvent (
    SDL_Event & e) [override], [virtual]
```

Implements [GameState](#).

## 4.2.2.5 render()

```
void ExitState::render () [override], [virtual]
```

Implements [GameState](#).

## 4.2.2.6 update()

```
void ExitState::update () [override], [virtual]
```

Implements [GameState](#).

## 4.2.3 Member Data Documentation

### 4.2.3.1 sExitState

```
ExitState ExitState::sExitState [static], [private]
```

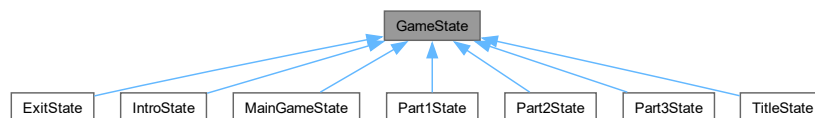
The documentation for this class was generated from the following files:

- src/include/states/[ExitState.h](#)
- src/states/[ExitState.cpp](#)

## 4.3 GameState Class Reference

```
#include <GameState.h>
```

Inheritance diagram for GameState:



### Public Member Functions

- virtual bool [enter](#) ()=0
- virtual bool [exit](#) ()=0
- virtual void [handleEvent](#) (SDL\_Event &e)=0
- virtual void [update](#) ()=0
- virtual void [render](#) ()=0
- virtual [~GameState](#) ()=default

### 4.3.1 Constructor & Destructor Documentation

#### 4.3.1.1 ~GameState()

```
virtual GameState::~GameState () [virtual], [default]
```

### 4.3.2 Member Function Documentation

#### 4.3.2.1 enter()

```
virtual bool GameState::enter () [pure virtual]
```

Implemented in [ExitState](#), [IntroState](#), [MainGameState](#), [Part1State](#), [Part2State](#), [Part3State](#), and [TitleState](#).



#### 4.3.2.2 exit()

```
virtual bool GameState::exit () [pure virtual]
```

Implemented in [ExitState](#), [IntroState](#), [MainGameState](#), [Part1State](#), [Part2State](#), [Part3State](#), and [TitleState](#).

#### 4.3.2.3 handleEvent()

```
virtual void GameState::handleEvent (  
    SDL_Event & e) [pure virtual]
```

Implemented in [ExitState](#), [IntroState](#), [MainGameState](#), [Part1State](#), [Part2State](#), [Part3State](#), and [TitleState](#).

#### 4.3.2.4 render()

```
virtual void GameState::render () [pure virtual]
```

Implemented in [ExitState](#), [IntroState](#), [MainGameState](#), [Part1State](#), [Part2State](#), [Part3State](#), and [TitleState](#).

#### 4.3.2.5 update()

```
virtual void GameState::update () [pure virtual]
```

Implemented in [ExitState](#), [IntroState](#), [MainGameState](#), [Part1State](#), [Part2State](#), [Part3State](#), and [TitleState](#).

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

- [src/include/states/GameState.h](#)

## 4.4 Global Class Reference

```
#include <Global.h>
```

### Static Public Attributes

- static const int [kScreenWidth](#) { [Map::PIXELS\\_PER\\_CELL](#) \* 15 }
- static const int [kScreenHeight](#) { [Map::PIXELS\\_PER\\_CELL](#) \* 11 }

### 4.4.1 Member Data Documentation

#### 4.4.1.1 kScreenHeight

```
const int Global::kScreenHeight { Map::PIXELS\_PER\_CELL * 11 } [static]
```

#### 4.4.1.2 kScreenWidth

```
const int Global::kScreenWidth { Map::PIXELS_PER_CELL * 15 } [static]
```

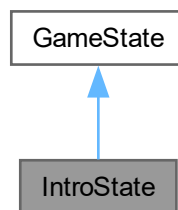
The documentation for this class was generated from the following file:

- [src/include/Global.h](#)

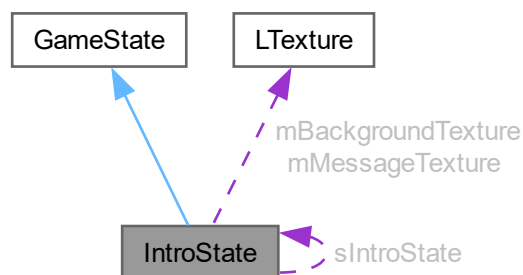
## 4.5 IntroState Class Reference

```
#include <IntroState.h>
```

Inheritance diagram for IntroState:



Collaboration diagram for IntroState:



### Public Member Functions

- bool [enter](#) () override
- bool [exit](#) () override
- void [handleEvent](#) (SDL\_Event &e) override
- void [update](#) () override
- void [render](#) () override

## Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

## Static Public Member Functions

- static [IntroState](#) \* [get](#) ()

## Private Member Functions

- [IntroState](#) ()

## Private Attributes

- [LTexture](#) mBackgroundTexture
- [LTexture](#) mMessageTexture

## Static Private Attributes

- static [IntroState](#) sIntroState

## 4.5.1 Constructor & Destructor Documentation

### 4.5.1.1 IntroState()

```
IntroState::IntroState () [private]
```

Here is the caller graph for this function:



## 4.5.2 Member Function Documentation

### 4.5.2.1 enter()

```
bool IntroState::enter () [override], [virtual]
```

Implements [GameState](#).

#### 4.5.2.2 exit()

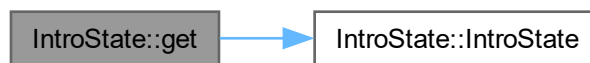
```
bool IntroState::exit () [override], [virtual]
```

Implements [GameState](#).

#### 4.5.2.3 get()

```
IntroState * IntroState::get () [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:

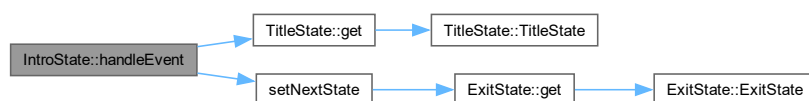


#### 4.5.2.4 handleEvent()

```
void IntroState::handleEvent (
    SDL_Event & e) [override], [virtual]
```

Implements [GameState](#).

Here is the call graph for this function:



#### 4.5.2.5 render()

```
void IntroState::render () [override], [virtual]
```

Implements [GameState](#).

#### 4.5.2.6 update()

```
void IntroState::update () [override], [virtual]
```

Implements [GameState](#).

### 4.5.3 Member Data Documentation

#### 4.5.3.1 mBackgroundTexture

```
LTexture IntroState::mBackgroundTexture [private]
```

#### 4.5.3.2 mMessageTexture

```
LTexture IntroState::mMessageTexture [private]
```

#### 4.5.3.3 sIntroState

```
IntroState IntroState::sIntroState [static], [private]
```

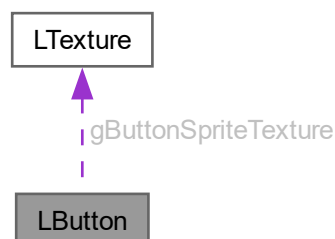
The documentation for this class was generated from the following files:

- [src/include/states/IntroState.h](#)
- [src/states/IntroState.cpp](#)

## 4.6 LButton Class Reference

```
#include <LButton.h>
```

Collaboration diagram for LButton:



## Public Member Functions

- [LButton](#) ()
- void [setPosition](#) (float x, float y)
- void [handleEvent](#) (SDL\_Event \*e)
- bool [setText](#) (const std::string &text, SDL\_Color textColor)
- void [render](#) ()
- bool [isClicked](#) () const

## Static Public Attributes

- static constexpr int [kButtonWidth](#) = 300
- static constexpr int [kButtonHeight](#) = 50

## Private Types

- enum [eButtonSprite](#) { [eButtonSpriteMouseOut](#) = 0 , [eButtonSpriteMouseOverMotion](#) = 1 , [eButtonSpriteMouseDown](#) = 2 , [eButtonSpriteMouseUp](#) = 3 }

## Private Attributes

- SDL\_FPoint [mPosition](#)
- [eButtonSprite](#) [mCurrentSprite](#)
- [LTexture](#) [gButtonSpriteTexture](#)

## 4.6.1 Member Enumeration Documentation

### 4.6.1.1 eButtonSprite

```
enum LButton::eButtonSprite [private]
```

#### Enumerator

<a href="#">eButtonSpriteMouseOut</a>	
<a href="#">eButtonSpriteMouseOverMotion</a>	
<a href="#">eButtonSpriteMouseDown</a>	
<a href="#">eButtonSpriteMouseUp</a>	

## 4.6.2 Constructor & Destructor Documentation

### 4.6.2.1 LButton()

```
LButton::LButton ()
```

## 4.6.3 Member Function Documentation

### 4.6.3.1 `handleEvent()`

```
void LButton::handleEvent (  
    SDL_Event * e)
```

### 4.6.3.2 `isClicked()`

```
bool LButton::isClicked () const
```

### 4.6.3.3 `render()`

```
void LButton::render ()
```

### 4.6.3.4 `setPosition()`

```
void LButton::setPosition (  
    float x,  
    float y)
```

### 4.6.3.5 `setText()`

```
bool LButton::setText (  
    const std::string & text,  
    SDL_Color textColor)
```

## 4.6.4 Member Data Documentation

### 4.6.4.1 `gButtonSpriteTexture`

```
LTexture LButton::gButtonSpriteTexture [private]
```

### 4.6.4.2 `kButtonHeight`

```
int LButton::kButtonHeight = 50 [static], [constexpr]
```

### 4.6.4.3 `kButtonWidth`

```
int LButton::kButtonWidth = 300 [static], [constexpr]
```

#### 4.6.4.4 mCurrentSprite

```
eButtonSprite LButton::mCurrentSprite [private]
```

#### 4.6.4.5 mPosition

```
SDL_FPoint LButton::mPosition [private]
```

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

- src/include/ui/LButton.h
- src/ui/LButton.cpp

## 4.7 LTexture Class Reference

```
#include <LTexture.h>
```

### Public Member Functions

- [LTexture](#) ()
- [~LTexture](#) ()
- bool [loadFromFile](#) (std::string path)
- bool [loadFromRenderedText](#) (std::string textureText, SDL\_Color textColor)
- void [destroy](#) ()
- void [setColor](#) (Uint8 r, Uint8 g, Uint8 b)
- void [setAlpha](#) (Uint8 alpha)
- void [setBlending](#) (SDL\_BlendMode blendMode)
- void [render](#) (float x, float y, SDL\_FRect \*clip=NULLPTR, float width=[kOriginalSize](#), float height=[kOriginalSize](#), double degrees=0.0, SDL\_FPoint \*center=NULLPTR, SDL\_FlipMode flipMode=SDL\_FLIP\_NONE)
- int [getWidth](#) ()
- int [getHeight](#) ()

### Static Public Attributes

- static constexpr float [kOriginalSize](#) = -1.f

### Private Attributes

- SDL\_Texture \* [mTexture](#)
- int [mWidth](#)
- int [mHeight](#)

### 4.7.1 Constructor & Destructor Documentation

#### 4.7.1.1 LTexture()

```
LTexture::LTexture ()
```



#### 4.7.1.2 ~LTexture()

```
LTexture::~~LTexture ()
```

Here is the call graph for this function:

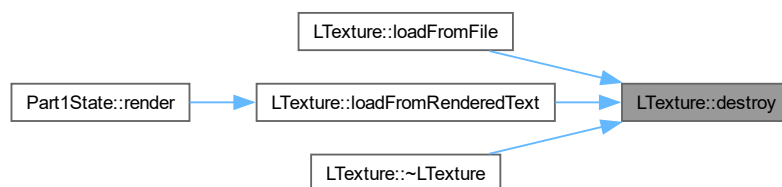


### 4.7.2 Member Function Documentation

#### 4.7.2.1 destroy()

```
void LTexture::destroy ()
```

Here is the caller graph for this function:



#### 4.7.2.2 getHeight()

```
int LTexture::getHeight ()
```

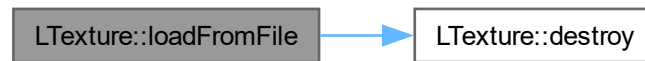
#### 4.7.2.3 getWidth()

```
int LTexture::getWidth ()
```

#### 4.7.2.4 loadFromFile()

```
bool LTexture::loadFromFile (  
    std::string path)
```

Here is the call graph for this function:



#### 4.7.2.5 loadFromRenderedText()

```
bool LTexture::loadFromRenderedText (  
    std::string textureText,  
    SDL_Color textColor)
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.2.6 render()

```
void LTexture::render (
    float x,
    float y,
    SDL_FRect * clip = nullptr,
    float width = kOriginalSize,
    float height = kOriginalSize,
    double degrees = 0.0,
    SDL_FPoint * center = nullptr,
    SDL_FlipMode flipMode = SDL_FLIP_NONE)
```

Here is the caller graph for this function:



#### 4.7.2.7 setAlpha()

```
void LTexture::setAlpha (
    Uint8 alpha)
```

#### 4.7.2.8 setBlending()

```
void LTexture::setBlending (
    SDL_BlendMode blendMode)
```

#### 4.7.2.9 setColor()

```
void LTexture::setColor (
    Uint8 r,
    Uint8 g,
    Uint8 b)
```

### 4.7.3 Member Data Documentation

#### 4.7.3.1 kOriginalSize

```
float LTexture::kOriginalSize = -1.f [static], [constexpr]
```

#### 4.7.3.2 mHeight

```
int LTexture::mHeight [private]
```

#### 4.7.3.3 mTexture

```
SDL_Texture* LTexture::mTexture [private]
```

#### 4.7.3.4 mWidth

```
int LTexture::mWidth [private]
```

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

- [src/include/ui/LTexture.h](#)
- [src/ui/LTexture.cpp](#)

## 4.8 LTimer Class Reference

```
#include <LTimer.h>
```

### Public Member Functions

- [LTimer \(\)](#)
- void [start \(\)](#)
- void [stop \(\)](#)
- void [pause \(\)](#)
- void [unpause \(\)](#)
- Uint64 [getTicksNS \(\)](#)
- bool [isStarted \(\)](#)
- bool [isPaused \(\)](#)

### Private Attributes

- Uint64 [mStartTicks](#)
- Uint64 [mPausedTicks](#)
- bool [mPaused](#)
- bool [mStarted](#)

## 4.8.1 Constructor & Destructor Documentation

### 4.8.1.1 LTimer()

```
LTimer::LTimer ()
```

## 4.8.2 Member Function Documentation

### 4.8.2.1 getTicksNS()

```
UInt64 LTimer::getTicksNS ()
```

Here is the caller graph for this function:



### 4.8.2.2 isPaused()

```
bool LTimer::isPaused ()
```

### 4.8.2.3 isStarted()

```
bool LTimer::isStarted ()
```

### 4.8.2.4 pause()

```
void LTimer::pause ()
```

### 4.8.2.5 start()

```
void LTimer::start ()
```

Here is the caller graph for this function:



#### 4.8.2.6 stop()

```
void LTimer::stop ()
```

#### 4.8.2.7 unpause()

```
void LTimer::unpause ()
```

### 4.8.3 Member Data Documentation

#### 4.8.3.1 mPaused

```
bool LTimer::mPaused [private]
```

#### 4.8.3.2 mPausedTicks

```
UInt64 LTimer::mPausedTicks [private]
```

#### 4.8.3.3 mStarted

```
bool LTimer::mStarted [private]
```

#### 4.8.3.4 mStartTicks

```
UInt64 LTimer::mStartTicks [private]
```

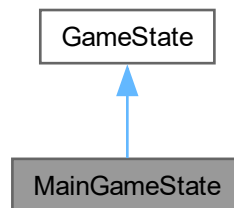
The documentation for this class was generated from the following files:

- [src/include/LTimer.h](#)
- [src/LTimer.cpp](#)

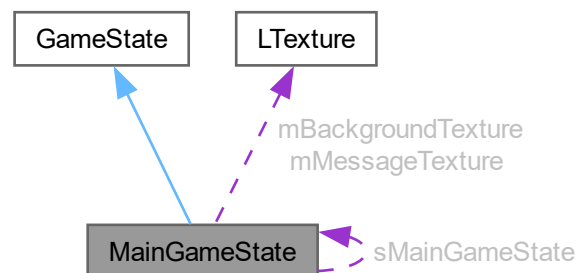
## 4.9 MainGameState Class Reference

```
#include <MainGameState.h>
```

Inheritance diagram for MainGameState:



Collaboration diagram for MainGameState:



### Public Member Functions

- bool [enter](#) () override
- bool [exit](#) () override
- void [handleEvent](#) (SDL\_Event &e) override
- void [update](#) () override
- void [render](#) () override

### Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

### Static Public Member Functions

- static [MainGameState](#) \* [get](#) ()

### Private Member Functions

- [MainGameState](#) ()

### Private Attributes

- [LTexture](#) [mBackgroundTexture](#)
- [LTexture](#) [mMessageTexture](#)

### Static Private Attributes

- static [MainGameState](#) [sMainGameState](#)

## 4.9.1 Constructor & Destructor Documentation

### 4.9.1.1 MainGameState()

```
MainGameState::MainGameState () [private]
```

Here is the caller graph for this function:



## 4.9.2 Member Function Documentation

### 4.9.2.1 enter()

```
bool MainGameState::enter () [override], [virtual]
```

Implements [GameState](#).

### 4.9.2.2 exit()

```
bool MainGameState::exit () [override], [virtual]
```

Implements [GameState](#).



#### 4.9.2.3 get()

```
MainGameState * MainGameState::get () [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.9.2.4 handleEvent()

```
void MainGameState::handleEvent (  
    SDL_Event & e) [override], [virtual]
```

Implements [GameState](#).

#### 4.9.2.5 render()

```
void MainGameState::render () [override], [virtual]
```

Implements [GameState](#).

#### 4.9.2.6 update()

```
void MainGameState::update () [override], [virtual]
```

Implements [GameState](#).

## 4.9.3 Member Data Documentation

### 4.9.3.1 mBackgroundTexture

`LTexture` `MainGameState::mBackgroundTexture` [private]

### 4.9.3.2 mMessageTexture

`LTexture` `MainGameState::mMessageTexture` [private]

### 4.9.3.3 sMainGameState

`MainGameState` `MainGameState::sMainGameState` [static], [private]

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

- `src/include/states/parts/MainGameState.h`
- `src/states/parts/MainGameState.cpp`

## 4.10 Map Class Reference

```
#include <Map.h>
```

### Classes

- struct `Cell`  
*Represents a single cell in the map grid.*

### Public Member Functions

- `Map` (`SDL_Renderer *renderer`, `int cellCountX`, `int cellCountY`)  
*Constructs a new `Map` object.*
- `void draw` (`SDL_Renderer *renderer`)  
*Draws all cells in the map.*
- `bool isCellWall` (`int x`, `int y`)  
*Checks if a cell is a wall.*
- `void setCellWall` (`int x`, `int y`, `bool setWall`)  
*Sets a cell's wall status.*
- `bool isTarget` (`int x`, `int y`)  
*Checks if a cell is a target.*
- `void setTarget` (`int x`, `int y`)  
*Sets a cell as the new target.*
- `bool isSpawner` (`int x`, `int y`)  
*Checks if a cell is a spawner.*
- `void setSpawner` (`int x`, `int y`)  
*Sets a cell as the new spawner.*
- `bool isValidPath` ()  
*Checks if a valid path exists from spawner to target.*
- `Vector2D getTargetPos` ()  
*Gets the position of the first target cell found.*
- `Vector2D getFlowNormal` (`int x`, `int y`)  
*Gets the normalized flow direction vector for a cell.*

### Static Public Attributes

- static constexpr int `PIXELS_PER_CELL` = 48  
*Number of pixels each cell occupies on screen.*

### Private Member Functions

- bool `isInbounds` (int x, int y)  
*Checks if a coordinate is within map boundaries.*
- void `drawCell` (SDL\_Renderer \*renderer, const `Cell` &cell)  
*Draws an individual cell.*
- void `calculateFlowField` ()  
*Recalculates the flow field for pathfinding.*
- void `calculateDistances` ()  
*Calculates distance values from target cells using BFS.*
- void `calculateFlowDirections` ()  
*Determines flow directions based on calculated distances.*

### Private Attributes

- std::vector< `Cell` > `cells`
- const int `cellCountX`
- const int `cellCountY`
- SDL\_Texture \* `textureCellWall`
- SDL\_Texture \* `textureCellTarget`
- SDL\_Texture \* `textureCellSpawner`
- SDL\_Texture \* `textureCellEmpty`
- SDL\_Texture \* `textureCellArrowUp`
- SDL\_Texture \* `textureCellArrowRight`
- SDL\_Texture \* `textureCellArrowDown`
- SDL\_Texture \* `textureCellArrowLeft`

### Static Private Attributes

- static const unsigned char `flowDistanceMax` = 255

## 4.10.1 Constructor & Destructor Documentation

### 4.10.1.1 Map()

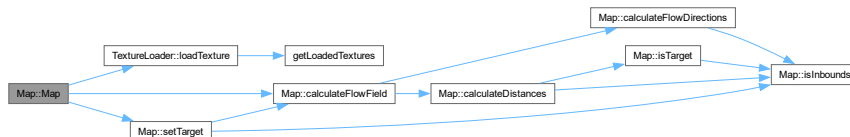
```
Map::Map (
    SDL_Renderer * renderer,
    int setCellCountX,
    int setCellCountY)
```

Constructs a new `Map` object.

## Parameters

<i>renderer</i>	The SDL renderer used for loading textures.
<i>setCellCountX</i>	The number of cells along the X-axis.
<i>setCellCountY</i>	The number of cells along the Y-axis.

Initializes cell textures, creates a grid of cells, sets the default target at the center, and calculates the initial flow field. Here is the call graph for this function:



## 4.10.2 Member Function Documentation

### 4.10.2.1 calculateDistances()

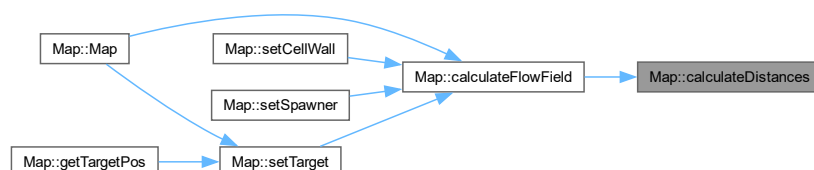
```
void Map::calculateDistances () [private]
```

Calculates distance values from target cells using BFS.

Initializes queue with target cell (distance 0), propagates distances to all reachable non-wall cells. Unreachable cells keep flowDistanceMax. Here is the call graph for this function:



Here is the caller graph for this function:

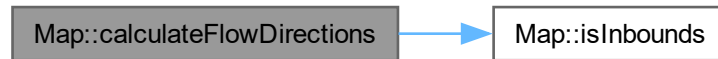


#### 4.10.2.2 calculateFlowDirections()

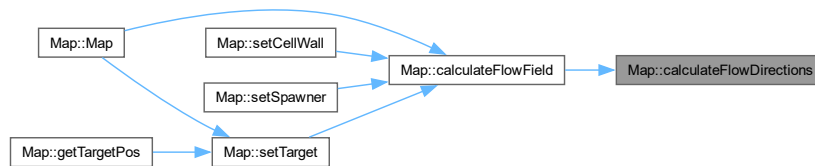
```
void Map::calculateFlowDirections () [private]
```

Determines flow directions based on calculated distances.

For each cell, sets flow direction towards the neighbor with the smallest distance value. Here is the call graph for this function:



Here is the caller graph for this function:

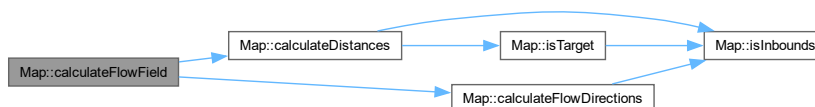


#### 4.10.2.3 calculateFlowField()

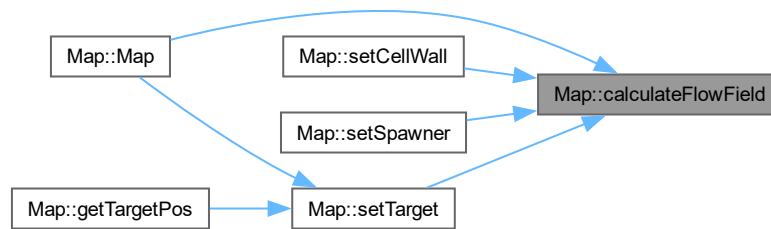
```
void Map::calculateFlowField () [private]
```

Recalculates the flow field for pathfinding.

Resets all flow data, calculates distances from targets using BFS, then determines optimal flow directions for each cell. Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.2.4 draw()

```
void Map::draw (
    SDL_Renderer * renderer)
```

Draws all cells in the map.

##### Parameters

<i>renderer</i>	The SDL renderer used for drawing.
-----------------	------------------------------------

Iterates through all cells and draws them using their respective textures based on type and flow direction. Here is the call graph for this function:



#### 4.10.2.5 drawCell()

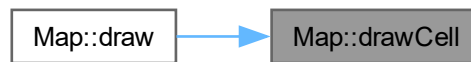
```
void Map::drawCell (
    SDL_Renderer * renderer,
    const Cell & cell) [private]
```

Draws an individual cell.

##### Parameters

<i>renderer</i>	The SDL renderer used for drawing.
<i>cell</i>	The cell to be drawn.

Selects the appropriate texture based on cell properties. Here is the caller graph for this function:



#### 4.10.2.6 getFlowNormal()

```
Vector2D Map::getFlowNormal (
    int x,
    int y)
```

Gets the normalized flow direction vector for a cell.

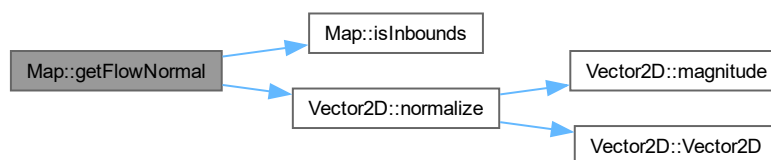
##### Parameters

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

##### Returns

**Vector2D** Normalized flow direction vector (zero vector if invalid).

Here is the call graph for this function:



#### 4.10.2.7 getTargetPos()

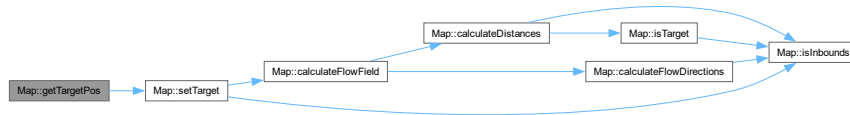
```
Vector2D Map::getTargetPos ()
```

Gets the position of the first target cell found.

Returns

**Vector2D** Position of the target in cell coordinates.

Falls back to center coordinates if no target found and sets one there. Here is the call graph for this function:



#### 4.10.2.8 isCellWall()

```
bool Map::isCellWall (
    int x,
    int y)
```

Checks if a cell is a wall.

Parameters

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

Returns

true If the cell is a wall.

false If the cell is not a wall or coordinates are invalid.

Here is the call graph for this function:



#### 4.10.2.9 isInbounds()

```
bool Map::isInbounds (
    int x,
    int y) [private]
```

Checks if a coordinate is within map boundaries.



**Parameters**

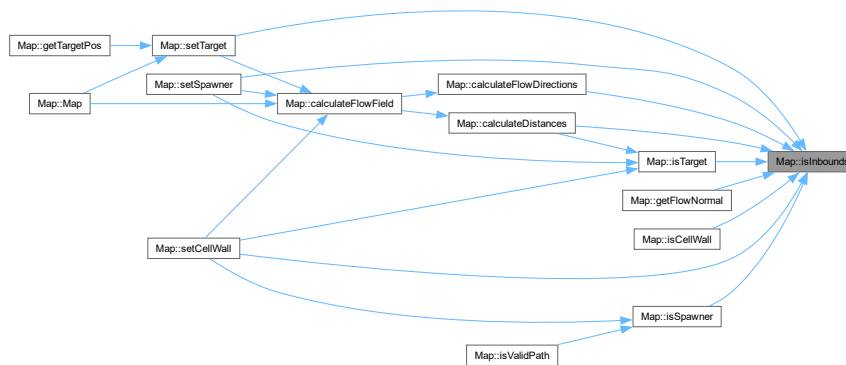
<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

**Returns**

true If the coordinates are within valid bounds.

false If the coordinates are out of bounds.

Here is the caller graph for this function:

**4.10.2.10 isSpawner()**

```

bool Map::isSpawner (
    int x,
    int y)

```

Checks if a cell is a spawner.

**Parameters**

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

**Returns**

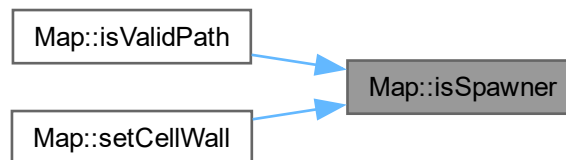
true If the cell is a spawner.

false If the cell is not a spawner or coordinates are invalid.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.2.11 isTarget()

```

bool Map::isTarget (
    int x,
    int y)
  
```

Checks if a cell is a target.

##### Parameters

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

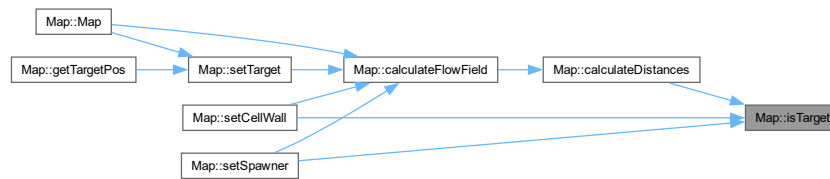
##### Returns

true If the cell is a target.  
false If the cell is not a target or coordinates are invalid.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.2.12 isValidPath()

```
bool Map::isValidPath ()
```

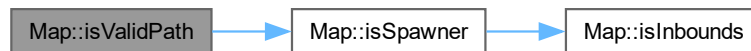
Checks if a valid path exists from spawner to target.

##### Returns

true If spawner exists and is reachable (valid flow path to target).

false If no spawner or spawner is unreachable.

Here is the call graph for this function:



#### 4.10.2.13 setCellWall()

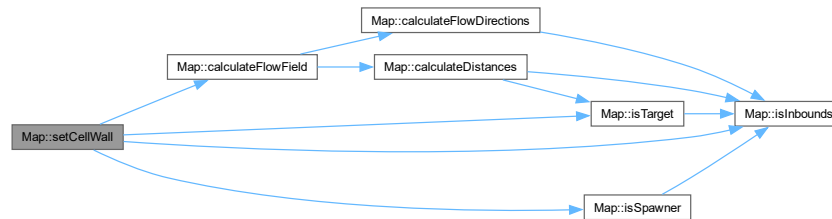
```
void Map::setCellWall (
    int x,
    int y,
    bool setWall)
```

Sets a cell's wall status.

##### Parameters

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.
<i>setWall</i>	true to make the cell a wall, false to clear.

Does nothing if coordinates are invalid or cell is target/spawner. Recalculates flow field after change. Here is the call graph for this function:



#### 4.10.2.14 setSpawner()

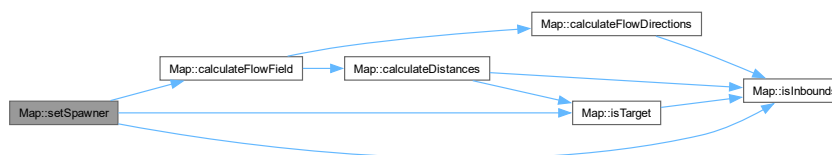
```
void Map::setSpawner (
    int x,
    int y)
```

Sets a cell as the new spawner.

##### Parameters

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

Clears existing spawners, sets new spawner, marks it non-wall, and recalculates flow field. Does nothing if coordinates are invalid or cell is a target. Here is the call graph for this function:



#### 4.10.2.15 setTarget()

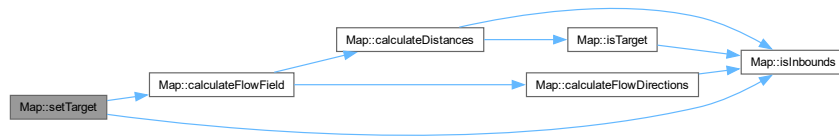
```
void Map::setTarget (
    int x,
    int y)
```

Sets a cell as the new target.

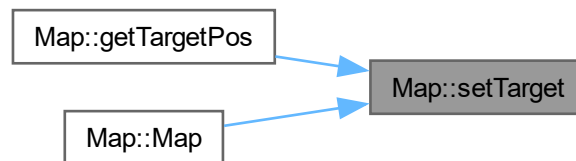
## Parameters

<i>x</i>	X-coordinate of the cell.
<i>y</i>	Y-coordinate of the cell.

Clears existing targets, sets new target, marks it non-wall/non-spawner, and recalculates flow field. Here is the call graph for this function:



Here is the caller graph for this function:



### 4.10.3 Member Data Documentation

#### 4.10.3.1 cellCountX

```
const int Map::cellCountX [private]
```

#### 4.10.3.2 cellCountY

```
const int Map::cellCountY [private]
```

Grid dimensions

#### 4.10.3.3 cells

```
std::vector<Cell> Map::cells [private]
```

Grid cells storage

#### 4.10.3.4 flowDistanceMax

```
const unsigned char Map::flowDistanceMax = 255 [static], [private]
```

#### 4.10.3.5 PIXELS\_PER\_CELL

```
int Map::PIXELS_PER_CELL = 48 [static], [constexpr]
```

Number of pixels each cell occupies on screen.

#### 4.10.3.6 textureCellArrowDown

```
SDL_Texture* Map::textureCellArrowDown [private]
```

Texture for downward flow

#### 4.10.3.7 textureCellArrowLeft

```
SDL_Texture* Map::textureCellArrowLeft [private]
```

Texture for leftward flow

#### 4.10.3.8 textureCellArrowRight

```
SDL_Texture* Map::textureCellArrowRight [private]
```

Texture for rightward flow

#### 4.10.3.9 textureCellArrowUp

```
SDL_Texture* Map::textureCellArrowUp [private]
```

Texture for upward flow

#### 4.10.3.10 textureCellEmpty

```
SDL_Texture* Map::textureCellEmpty [private]
```

Texture for empty cells

#### 4.10.3.11 textureCellSpawner

```
SDL_Texture* Map::textureCellSpawner [private]
```

Texture for spawner cells

#### 4.10.3.12 textureCellTarget

```
SDL_Texture* Map::textureCellTarget [private]
```

Texture for target cells

#### 4.10.3.13 textureCellWall

```
SDL_Texture* Map::textureCellWall [private]
```

Texture for wall cells

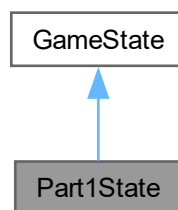
The documentation for this class was generated from the following files:

- [src/include/map/Map.h](#)
- [src/map/Map.cpp](#)

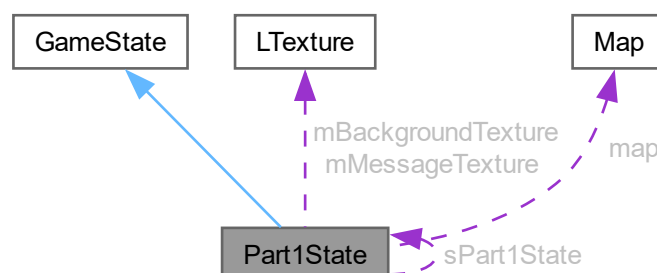
## 4.11 Part1State Class Reference

```
#include <Part1State.h>
```

Inheritance diagram for Part1State:



Collaboration diagram for Part1State:



### Public Member Functions

- bool [enter](#) () override  
*Initialize the state.*
- bool [exit](#) () override  
*Clean up the state.*
- void [handleEvent](#) (SDL\_Event &e) override  
*Handle input events.*
- void [update](#) () override  
*Update the state.*
- void [render](#) () override  
*Render the state.*

### Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

### Static Public Member Functions

- static [Part1State](#) \* [get](#) ()  
*Get the singleton instance of [Part1State](#).*

### Private Member Functions

- [Part1State](#) ()  
*Default constructor for [Part1State](#).*

### Private Attributes

- [LTexture](#) [mBackgroundTexture](#)
- [LTexture](#) [mMessageTexture](#)
- int [mouseDownStatus](#) = 0
- int [keyDownStatus](#) = 0
- [Map](#) \* [map](#) = nullptr  
*Pointer to the map being edited.*

### Static Private Attributes

- static [Part1State](#) [sPart1State](#)  
*Singleton instance of the [Part1State](#).*



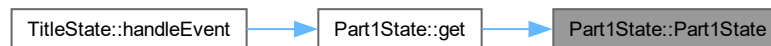
## 4.11.1 Constructor & Destructor Documentation

### 4.11.1.1 Part1State()

```
Part1State::Part1State () [private]
```

Default constructor for [Part1State](#).

Here is the caller graph for this function:



## 4.11.2 Member Function Documentation

### 4.11.2.1 enter()

```
bool Part1State::enter () [override], [virtual]
```

Initialize the state.

Creates a new [Map](#) instance sized to fit the screen dimensions

Returns

true if initialization was successful

Implements [GameState](#).

### 4.11.2.2 exit()

```
bool Part1State::exit () [override], [virtual]
```

Clean up the state.

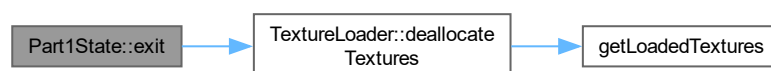
Deallocates textures and frees the map

Returns

true if cleanup was successful

Implements [GameState](#).

Here is the call graph for this function:



#### 4.11.2.3 `get()`

```
Part1State * Part1State::get () [static]
```

Get the singleton instance of [Part1State](#).

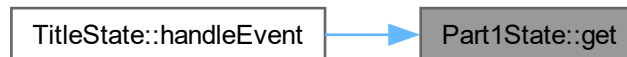
##### Returns

Pointer to the [Part1State](#) singleton

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.2.4 `handleEvent()`

```
void Part1State::handleEvent (  
    SDL_Event & e) [override], [virtual]
```

Handle input events.

Processes mouse and keyboard input to:

- Place/remove walls with left/right click
- Set target/spawner with Shift + left/right click

##### Parameters

<i>e</i>	SDL event to process
----------	----------------------

Implements [GameState](#).

#### 4.11.2.5 render()

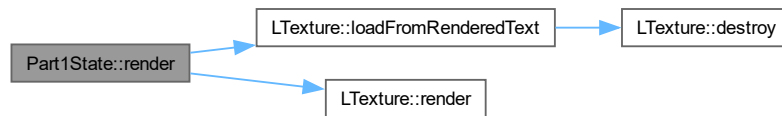
```
void Part1State::render () [override], [virtual]
```

Render the state.

Draws the map and displays path validity status in the top-left corner

Implements [GameState](#).

Here is the call graph for this function:



#### 4.11.2.6 update()

```
void Part1State::update () [override], [virtual]
```

Update the state.

Currently empty, reserved for future implementation

Implements [GameState](#).

### 4.11.3 Member Data Documentation

#### 4.11.3.1 keyDownStatus

```
int Part1State::keyDownStatus = 0 [private]
```

#### 4.11.3.2 map

```
Map* Part1State::map = nullptr [private]
```

Pointer to the map being edited.

nullptr if no map is currently loaded

#### 4.11.3.3 mBackgroundTexture

```
LTexture Part1State::mBackgroundTexture [private]
```

#### 4.11.3.4 mMessageTexture

```
LTexture Part1State::mMessageTexture [private]
```

#### 4.11.3.5 mouseDownStatus

```
int Part1State::mouseDownStatus = 0 [private]
```

#### 4.11.3.6 sPart1State

```
Part1State Part1State::sPart1State [static], [private]
```

Singleton instance of the [Part1State](#).

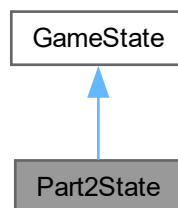
The documentation for this class was generated from the following files:

- [src/include/states/parts/Part1State.h](#)
- [src/states/parts/Part1State.cpp](#)

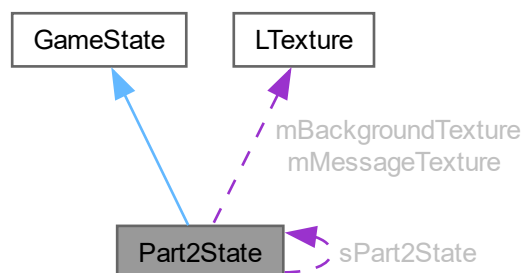
## 4.12 Part2State Class Reference

```
#include <Part2State.h>
```

Inheritance diagram for Part2State:



Collaboration diagram for Part2State:



### Public Member Functions

- bool [enter](#) () override
- bool [exit](#) () override
- void [handleEvent](#) (SDL\_Event &e) override
- void [update](#) () override
- void [render](#) () override

### Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

### Static Public Member Functions

- static [Part2State](#) \* [get](#) ()

### Private Member Functions

- [Part2State](#) ()

### Private Attributes

- [LTexture](#) [mBackgroundTexture](#)
- [LTexture](#) [mMessageTexture](#)

### Static Private Attributes

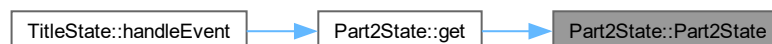
- static [Part2State](#) [sPart2State](#)

## 4.12.1 Constructor & Destructor Documentation

### 4.12.1.1 [Part2State](#)()

```
Part2State::Part2State () [private]
```

Here is the caller graph for this function:



## 4.12.2 Member Function Documentation

### 4.12.2.1 enter()

```
bool Part2State::enter () [override], [virtual]
```

Implements [GameState](#).

### 4.12.2.2 exit()

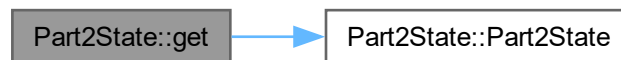
```
bool Part2State::exit () [override], [virtual]
```

Implements [GameState](#).

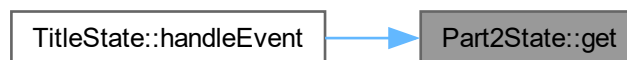
### 4.12.2.3 get()

```
Part2State * Part2State::get () [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.12.2.4 handleEvent()

```
void Part2State::handleEvent (  
    SDL_Event & e) [override], [virtual]
```

Implements [GameState](#).

#### 4.12.2.5 render()

```
void Part2State::render () [override], [virtual]
```

Implements [GameState](#).

#### 4.12.2.6 update()

```
void Part2State::update () [override], [virtual]
```

Implements [GameState](#).

### 4.12.3 Member Data Documentation

#### 4.12.3.1 mBackgroundTexture

```
LTexture Part2State::mBackgroundTexture [private]
```

#### 4.12.3.2 mMessageTexture

```
LTexture Part2State::mMessageTexture [private]
```

#### 4.12.3.3 sPart2State

```
Part2State Part2State::sPart2State [static], [private]
```

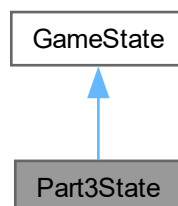
The documentation for this class was generated from the following files:

- `src/include/states/parts/Part2State.h`
- `src/states/parts/Part2State.cpp`

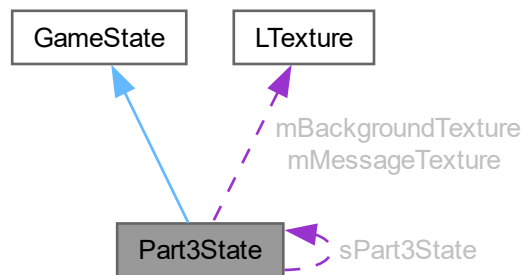
## 4.13 Part3State Class Reference

```
#include <Part3State.h>
```

Inheritance diagram for Part3State:



Collaboration diagram for Part3State:



### Public Member Functions

- bool [enter](#) () override
- bool [exit](#) () override
- void [handleEvent](#) (SDL\_Event &e) override
- void [update](#) () override
- void [render](#) () override

### Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

### Static Public Member Functions

- static [Part3State](#) \* [get](#) ()

### Private Member Functions

- [Part3State](#) ()

### Private Attributes

- [LTexture](#) mBackgroundTexture
- [LTexture](#) mMessageTexture

### Static Private Attributes

- static [Part3State](#) sPart3State

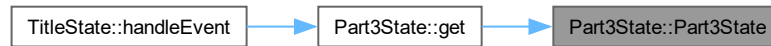


## 4.13.1 Constructor & Destructor Documentation

### 4.13.1.1 Part3State()

```
Part3State::Part3State () [private]
```

Here is the caller graph for this function:



## 4.13.2 Member Function Documentation

### 4.13.2.1 enter()

```
bool Part3State::enter () [override], [virtual]
```

Implements [GameState](#).

### 4.13.2.2 exit()

```
bool Part3State::exit () [override], [virtual]
```

Implements [GameState](#).

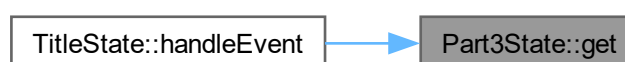
### 4.13.2.3 get()

```
Part3State * Part3State::get () [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.2.4 `handleEvent()`

```
void Part3State::handleEvent (
    SDL_Event & e) [override], [virtual]
```

Implements [GameState](#).

#### 4.13.2.5 `render()`

```
void Part3State::render () [override], [virtual]
```

Implements [GameState](#).

#### 4.13.2.6 `update()`

```
void Part3State::update () [override], [virtual]
```

Implements [GameState](#).

### 4.13.3 Member Data Documentation

#### 4.13.3.1 `mBackgroundTexture`

```
LTexture Part3State::mBackgroundTexture [private]
```

#### 4.13.3.2 `mMessageTexture`

```
LTexture Part3State::mMessageTexture [private]
```

#### 4.13.3.3 `sPart3State`

```
Part3State Part3State::sPart3State [static], [private]
```

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

- `src/include/states/parts/Part3State.h`
- `src/states/parts/Part3State.cpp`

## 4.14 TextureLoader Class Reference

```
#include <TextureLoader.h>
```

### Static Public Member Functions

- static SDL\_Texture \* [loadTexture](#) (SDL\_Renderer \*renderer, std::string filename)  
*Load a texture from a file and return it.*
- static void [deallocateTextures](#) ()  
*Destroy all loaded textures and clear the texture cache.*

### Static Private Attributes

- static std::unordered\_map< std::string, SDL\_Texture \* > [loadedTextures](#)  
*Static map storing all loaded textures.*
- static const std::string [TEXTURE\\_PATH](#) = "assets/"

## 4.14.1 Member Function Documentation

### 4.14.1.1 deallocateTextures()

```
void TextureLoader::deallocateTextures () [static]
```

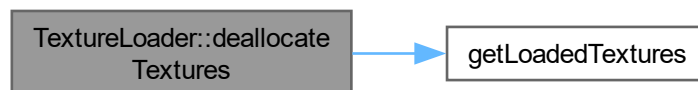
Destroy all loaded textures and clear the texture cache.

This function should be called during cleanup to prevent memory leaks. It iterates through all cached textures, properly destroys each one using SDL, and clears the texture map.

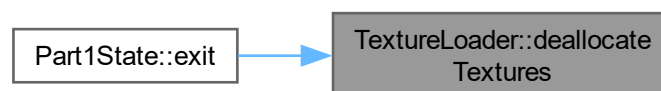
#### Warning

After calling this function, any existing pointers to previously loaded textures will be invalid

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.1.2 loadTexture()

```
SDL_Texture * TextureLoader::loadTexture (
    SDL_Renderer * renderer,
    std::string filename) [static]
```

Load a texture from a file and return it.

This function implements texture caching. If the requested texture has already been loaded, it returns the cached version instead of loading it again. New textures are automatically added to the cache.

##### Parameters

<i>renderer</i>	The SDL renderer used to create the texture
<i>filename</i>	The path to the texture file, relative to the 'assets' folder

##### Returns

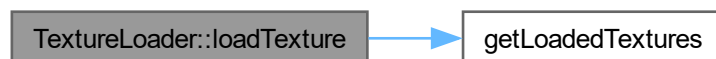
SDL\_Texture\* The loaded texture, or nullptr if loading failed

##### Note

The returned texture has SDL\_BLENDMODE\_BLEND enabled by default

The function automatically manages the temporary SDL\_Surface used during loading

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.14.2 Member Data Documentation

### 4.14.2.1 loadedTextures

```
std::unordered_map< std::string, SDL_Texture * > TextureLoader::loadedTextures [static],  
[private]
```

Static map storing all loaded textures.

### 4.14.2.2 TEXTURE\_PATH

```
const std::string TextureLoader::TEXTURE_PATH = "assets/" [inline], [static], [private]
```

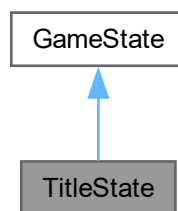
The documentation for this class was generated from the following files:

- src/include/util/[TextureLoader.h](#)
- src/util/[TextureLoader.cpp](#)

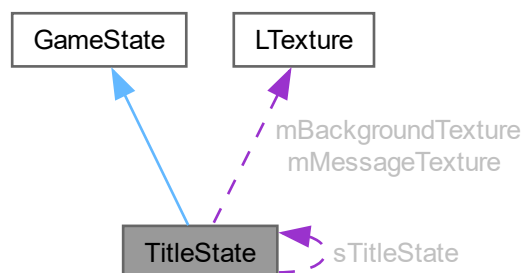
## 4.15 TitleState Class Reference

```
#include <TitleState.h>
```

Inheritance diagram for TitleState:



Collaboration diagram for TitleState:



### Public Member Functions

- bool [enter](#) () override
- bool [exit](#) () override
- void [handleEvent](#) (SDL\_Event &e) override
- void [update](#) () override
- void [render](#) () override

### Public Member Functions inherited from [GameState](#)

- virtual [~GameState](#) ()=default

### Static Public Member Functions

- static [TitleState](#) \* [get](#) ()

### Private Member Functions

- [TitleState](#) ()

### Private Attributes

- [LTexture](#) [mBackgroundTexture](#)
- [LTexture](#) [mMessageTexture](#)

### Static Private Attributes

- static [TitleState](#) [sTitleState](#)

## 4.15.1 Constructor & Destructor Documentation

### 4.15.1.1 [TitleState](#)()

`TitleState::TitleState () [private]`

Here is the caller graph for this function:



## 4.15.2 Member Function Documentation

### 4.15.2.1 enter()

```
bool TitleState::enter () [override], [virtual]
```

Implements [GameState](#).

### 4.15.2.2 exit()

```
bool TitleState::exit () [override], [virtual]
```

Implements [GameState](#).

### 4.15.2.3 get()

```
TitleState * TitleState::get () [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:

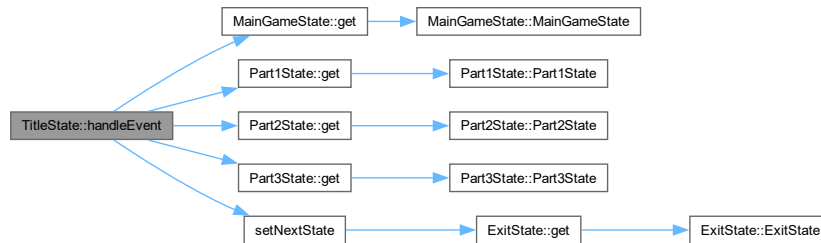


#### 4.15.2.4 `handleEvent()`

```
void TitleState::handleEvent (
    SDL_Event & e) [override], [virtual]
```

Implements [GameState](#).

Here is the call graph for this function:



#### 4.15.2.5 `render()`

```
void TitleState::render () [override], [virtual]
```

Implements [GameState](#).

#### 4.15.2.6 `update()`

```
void TitleState::update () [override], [virtual]
```

Implements [GameState](#).

### 4.15.3 Member Data Documentation

#### 4.15.3.1 `mBackgroundTexture`

```
LTexture TitleState::mBackgroundTexture [private]
```

#### 4.15.3.2 `mMessageTexture`

```
LTexture TitleState::mMessageTexture [private]
```



### 4.15.3.3 sTitleState

```
TitleState TitleState::sTitleState [static], [private]
```

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

- src/include/states/[TitleState.h](#)
- src/states/[TitleState.cpp](#)

## 4.16 Vector2D Class Reference

```
#include <Vector2D.h>
```

### Public Member Functions

- [Vector2D](#) (float setX, float setY)
- [Vector2D](#) (const [Vector2D](#) &other)
- [Vector2D](#) (float angleRad)
- [Vector2D](#) ()
- float [angle](#) ()
- float [magnitude](#) ()
- [Vector2D](#) [normalize](#) ()
- [Vector2D](#) [getNegativeReciprocal](#) ()
- float [dot](#) (const [Vector2D](#) &other)
- float [cross](#) (const [Vector2D](#) &other)
- float [angleBetween](#) (const [Vector2D](#) &other)
- [Vector2D](#) [operator+](#) (const float amount)
- [Vector2D](#) [operator-](#) (const float amount)
- [Vector2D](#) [operator\\*](#) (const float amount)
- [Vector2D](#) [operator/](#) (const float amount)
- [Vector2D](#) [operator+](#) (const [Vector2D](#) &other)
- [Vector2D](#) [operator-](#) (const [Vector2D](#) &other)
- [Vector2D](#) [operator\\*](#) (const [Vector2D](#) &other)
- [Vector2D](#) [operator/](#) (const [Vector2D](#) &other)
- [Vector2D](#) & [operator+=](#) (const float amount)
- [Vector2D](#) & [operator-=](#) (const float amount)
- [Vector2D](#) & [operator\\*=](#) (const float amount)
- [Vector2D](#) & [operator/=](#) (const float amount)
- [Vector2D](#) & [operator+=](#) (const [Vector2D](#) &other)
- [Vector2D](#) & [operator-=](#) (const [Vector2D](#) &other)
- [Vector2D](#) & [operator\\*=](#) (const [Vector2D](#) &other)
- [Vector2D](#) & [operator/=](#) (const [Vector2D](#) &other)

### Public Attributes

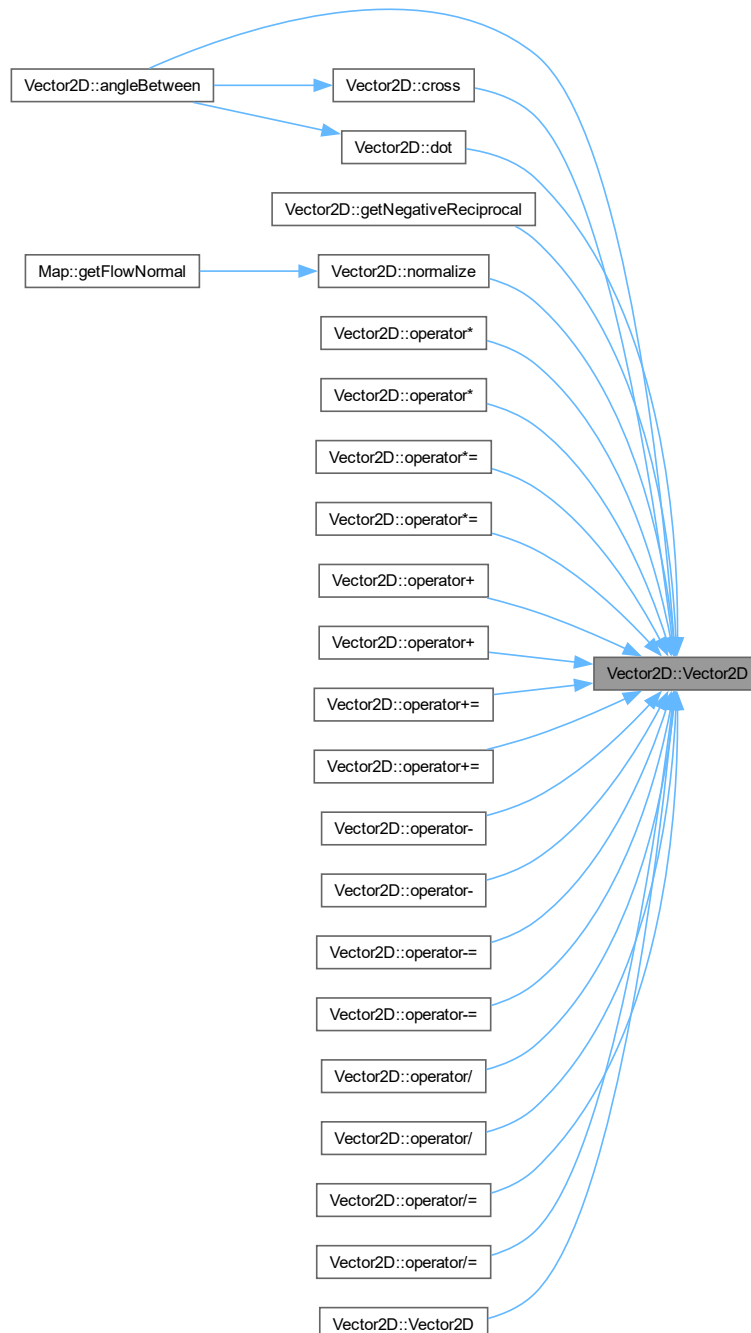
- float [x](#)
- float [y](#)

## 4.16.1 Constructor & Destructor Documentation

### 4.16.1.1 Vector2D() [1/4]

```
Vector2D::Vector2D (
    float setX,
    float setY) [inline]
```

Here is the caller graph for this function:



#### 4.16.1.2 Vector2D() [2/4]

```
Vector2D::Vector2D (  
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



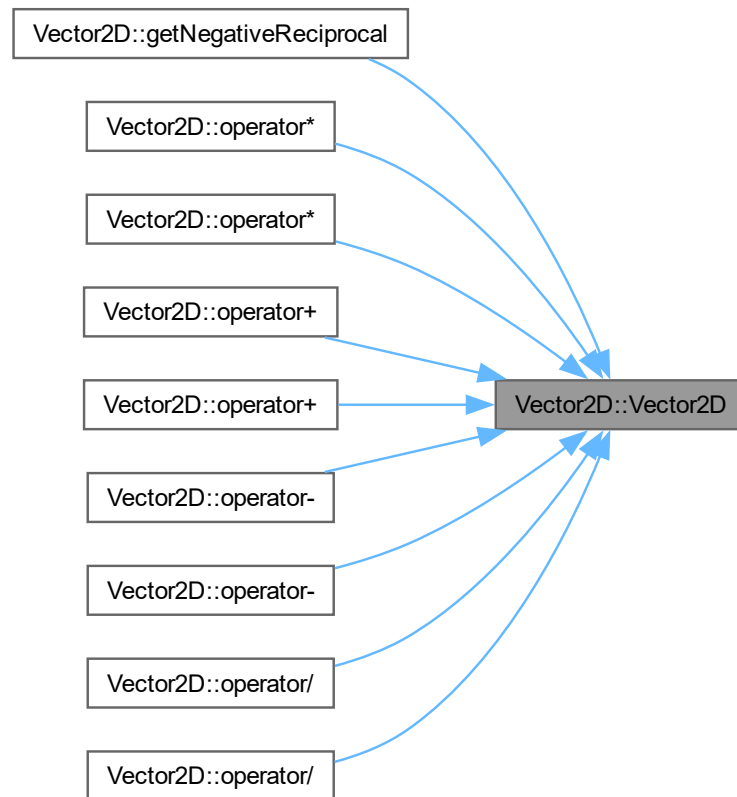
#### 4.16.1.3 Vector2D() [3/4]

```
Vector2D::Vector2D (  
    float angleRad) [inline]
```

#### 4.16.1.4 Vector2D() [4/4]

```
Vector2D::Vector2D () [inline]
```

Here is the caller graph for this function:



## 4.16.2 Member Function Documentation

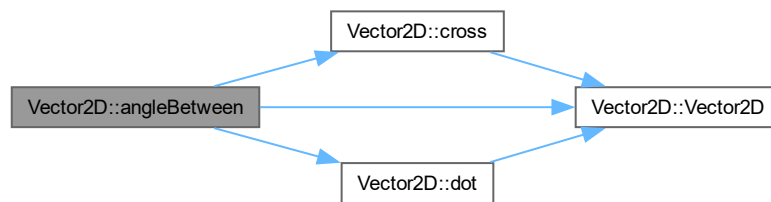
### 4.16.2.1 `angle()`

```
float Vector2D::angle () [inline]
```

### 4.16.2.2 `angleBetween()`

```
float Vector2D::angleBetween (
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



#### 4.16.2.3 cross()

```
float Vector2D::cross (  
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



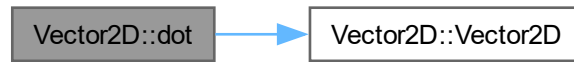
Here is the caller graph for this function:



#### 4.16.2.4 dot()

```
float Vector2D::dot (  
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



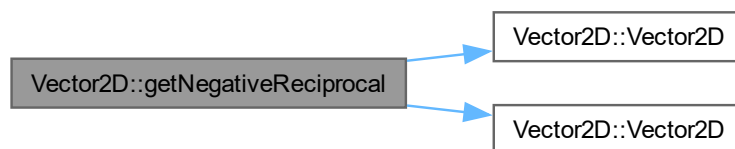
Here is the caller graph for this function:



#### 4.16.2.5 `getNegativeReciprocal()`

```
Vector2D Vector2D::getNegativeReciprocal () [inline]
```

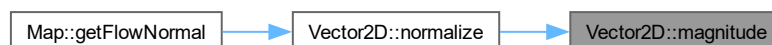
Here is the call graph for this function:



#### 4.16.2.6 `magnitude()`

```
float Vector2D::magnitude () [inline]
```

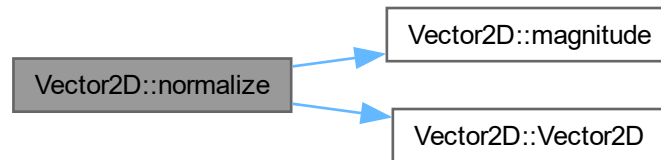
Here is the caller graph for this function:



#### 4.16.2.7 normalize()

```
Vector2D Vector2D::normalize ()
```

Here is the call graph for this function:



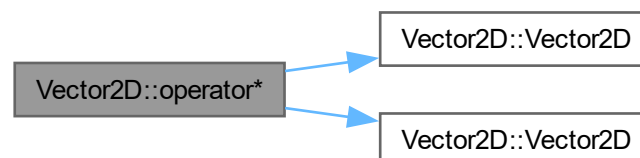
Here is the caller graph for this function:



#### 4.16.2.8 operator\*() [1/2]

```
Vector2D Vector2D::operator* (
    const float amount) [inline]
```

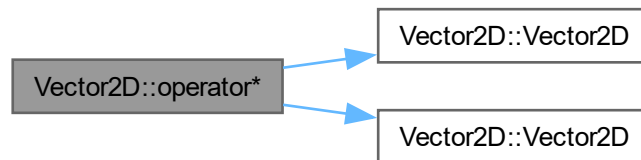
Here is the call graph for this function:



#### 4.16.2.9 `operator*()` [2/2]

```
Vector2D Vector2D::operator* (
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



#### 4.16.2.10 `operator*=(1)` [1/2]

```
Vector2D & Vector2D::operator*= (
    const float amount) [inline]
```

Here is the call graph for this function:



#### 4.16.2.11 `operator*=(2)` [2/2]

```
Vector2D & Vector2D::operator*= (
    const Vector2D & other) [inline]
```

Here is the call graph for this function:

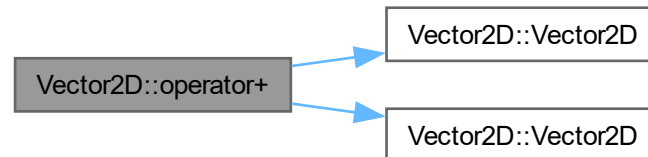




#### 4.16.2.12 operator+() [1/2]

```
Vector2D Vector2D::operator+ (  
    const float amount) [inline]
```

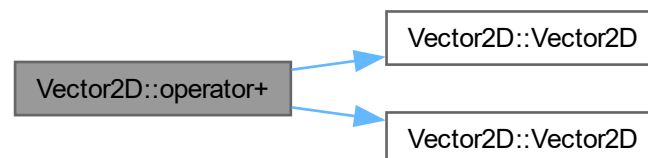
Here is the call graph for this function:



#### 4.16.2.13 operator+() [2/2]

```
Vector2D Vector2D::operator+ (  
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



#### 4.16.2.14 operator+=() [1/2]

```
Vector2D & Vector2D::operator+= (  
    const float amount) [inline]
```

Here is the call graph for this function:



#### 4.16.2.15 operator+=() [2/2]

```
Vector2D & Vector2D::operator+= (  
    const Vector2D & other) [inline]
```

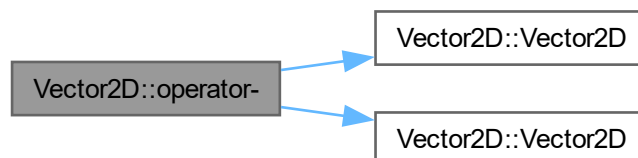
Here is the call graph for this function:



#### 4.16.2.16 operator-() [1/2]

```
Vector2D Vector2D::operator- (  
    const float amount) [inline]
```

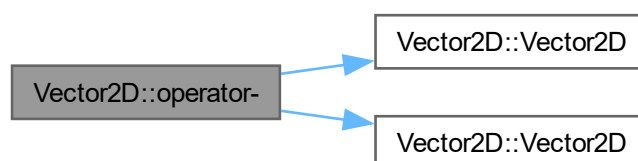
Here is the call graph for this function:



#### 4.16.2.17 operator-() [2/2]

```
Vector2D Vector2D::operator- (  
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



**4.16.2.18 operator-=() [1/2]**

```
Vector2D & Vector2D::operator-= (  
    const float amount) [inline]
```

Here is the call graph for this function:

**4.16.2.19 operator-=() [2/2]**

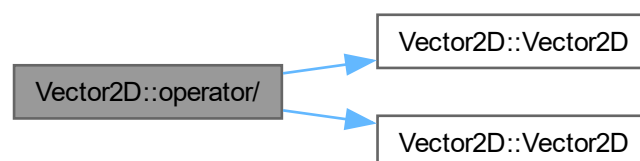
```
Vector2D & Vector2D::operator-= (  
    const Vector2D & other) [inline]
```

Here is the call graph for this function:

**4.16.2.20 operator/() [1/2]**

```
Vector2D Vector2D::operator/ (  
    const float amount) [inline]
```

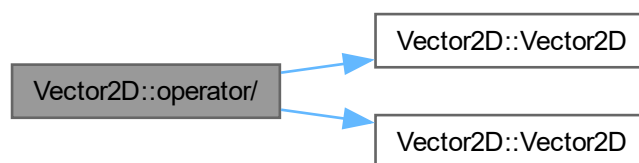
Here is the call graph for this function:



#### 4.16.2.21 operator/() [2/2]

```
Vector2D Vector2D::operator/ (
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



#### 4.16.2.22 operator/=() [1/2]

```
Vector2D & Vector2D::operator/= (
    const float amount) [inline]
```

Here is the call graph for this function:



#### 4.16.2.23 operator/=() [2/2]

```
Vector2D & Vector2D::operator/= (
    const Vector2D & other) [inline]
```

Here is the call graph for this function:



### 4.16.3 Member Data Documentation

#### 4.16.3.1 x

```
float Vector2D::x
```

#### 4.16.3.2 y

```
float Vector2D::y
```

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

- [src/include/util/Vector2D.h](#)
- [src/util/Vector2D.cpp](#)



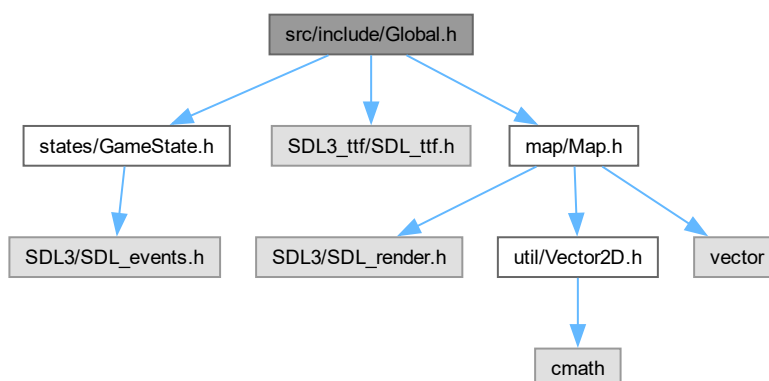
## Chapter 5

# File Documentation

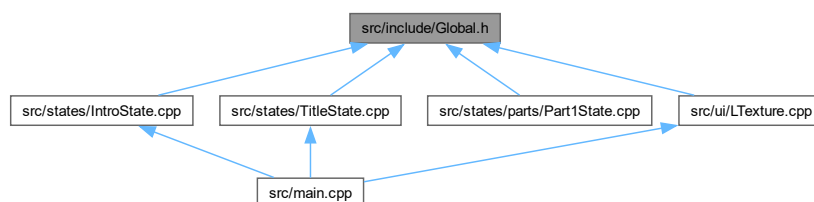
### 5.1 src/include/Global.h File Reference

```
#include <states/GameState.h>
#include <SDL3_ttf/SDL_ttf.h>
#include <map/Map.h>
```

Include dependency graph for Global.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Global](#)

## Functions

- bool [init](#) ()
- bool [loadMedia](#) ()
- void [close](#) ()
- void [setNextState](#) ([GameState](#) \*nextState)
- void [changeState](#) ()

## Variables

- SDL\_Renderer \* [gRenderer](#)
- TTF\_Font \* [gFont](#)

### 5.1.1 Function Documentation

#### 5.1.1.1 [changeState\(\)](#)

```
void changeState ()
```

Here is the caller graph for this function:



#### 5.1.1.2 [close\(\)](#)

```
void close ()
```

Here is the caller graph for this function:





### 5.1.1.3 init()

```
bool init ()
```

Here is the caller graph for this function:



### 5.1.1.4 loadMedia()

```
bool loadMedia ()
```

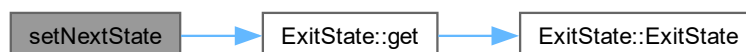
Here is the caller graph for this function:



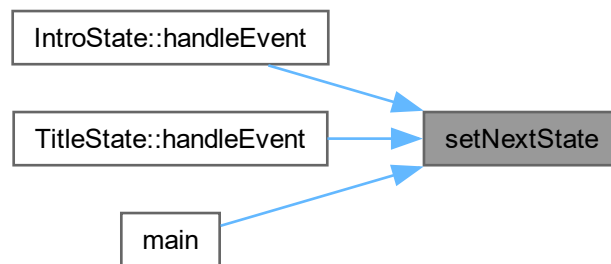
### 5.1.1.5 setNextState()

```
void setNextState (  
    GameState * nextState)
```

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.1.2 Variable Documentation

### 5.1.2.1 gFont

```
TTF_Font* gFont [extern]
```

### 5.1.2.2 gRenderer

```
SDL_Renderer* gRenderer [extern]
```

## 5.2 Global.h

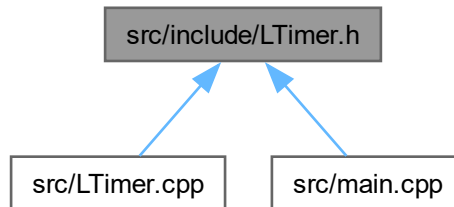
[Go to the documentation of this file.](#)

```

00001 #ifndef GLOBAL_H
00002 #define GLOBAL_H
00003
00004 #include <states/GameState.h>
00005 #include <SDL3_ttf/SDL_ttf.h>
00006 #include <map/Map.h>
00007
00008 extern SDL_Renderer* gRenderer;
00009 extern TTF_Font* gFont;
00010
00011 class Global {
00012 public:
00013     //Screen dimension constants
00014     static const int kScreenWidth{ Map::PIXELS_PER_CELL * 15 };
00015     static const int kScreenHeight{ Map::PIXELS_PER_CELL * 11 };
00016 };
00017
00018 /* Function Prototypes */
00019 //Starts up SDL and creates window
00020 bool init();
00021
00022 //Loads media
00023 bool loadMedia();
00024
00025 //Frees media and shuts down SDL
00026 void close();
00027
00028 //State managers
00029 void setNextState(GameState* nextState);
00030 void changeState();
00031 #endif
  
```

## 5.3 src/include/LTimer.h File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- class [LTimer](#)

## 5.4 LTimer.h

[Go to the documentation of this file.](#)

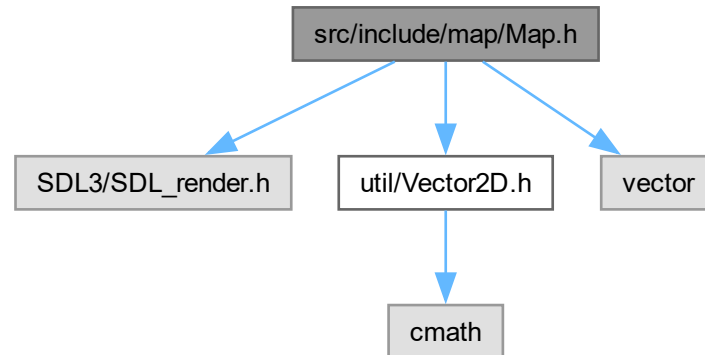
```

00001 class LTimer {
00002 public:
00003     //Initializes variables
00004     LTimer();
00005
00006     //The various clock actions
00007     void start();
00008     void stop();
00009     void pause();
00010     void unpause();
00011
00012     //Gets the timer's time
00013     Uint64 getTicksNS();
00014
00015     //Checks the status of the timer
00016     bool isStarted();
00017     bool isPaused();
00018
00019 private:
00020     //The clock time when the timer started
00021     Uint64 mStartTicks;
00022
00023     //The ticks stored when the timer was paused
00024     Uint64 mPausedTicks;
00025
00026     //The timer status
00027     bool mPaused;
00028     bool mStarted;
00029 };
  
```

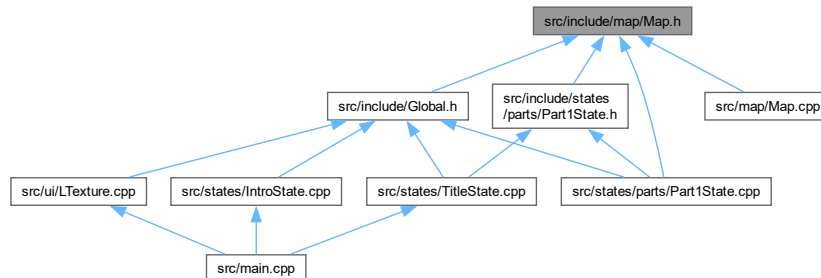
## 5.5 src/include/map/Map.h File Reference

Responsible for generating the tower defense map.

```
#include <SDL3/SDL_render.h>
#include <util/Vector2D.h>
#include <vector>
Include dependency graph for Map.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [Map](#)
- struct [Map::Cell](#)

*Represents a single cell in the map grid.*

## 5.5.1 Detailed Description

Responsible for generating the tower defense map.

Author

Nathan Grenier

## Date

2025-02-15

A class that manages a grid-based map for a tower defense game. The map consists of cells that can be walls, spawners, or targets. It also implements a flow field pathfinding system to guide enemies from spawners to targets.

## 5.6 Map.h

[Go to the documentation of this file.](#)

```

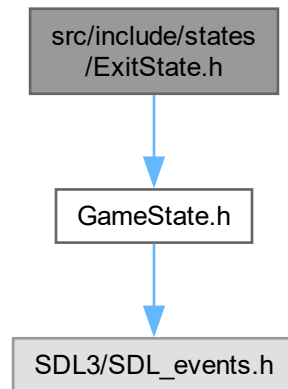
00001
00012 #pragma once
00013
00014 #include <SDL3/SDL_render.h>
00015 #include <util/Vector2D.h>
00016 #include <vector>
00017
00018 class Map {
00019 private:
00020     static const unsigned char flowDistanceMax = 255;
00021
00022     struct Cell {
00023         int x = 0;
00024         int y = 0;
00025         bool isWall = false;
00026         bool isSpawner = false;
00027         bool isTarget = false;
00028         int flowDirectionX = 0;
00029         int flowDirectionY = 0;
00030         unsigned char flowDistance = flowDistanceMax;
00031     };
00032
00033 public:
00034     static constexpr int PIXELS_PER_CELL = 48;
00035
00036     Map(SDL_Renderer* renderer, int cellCountX, int cellCountY);
00037     void draw(SDL_Renderer* renderer);
00038
00039     bool isCellWall(int x, int y);
00040     void setCellWall(int x, int y, bool setWall);
00041     bool isTarget(int x, int y);
00042     void setTarget(int x, int y);
00043     bool isSpawner(int x, int y);
00044     void setSpawner(int x, int y);
00045     bool isValidPath();
00046     Vector2D getTargetPos();
00047     Vector2D getFlowNormal(int x, int y);
00048
00049 private:
00050     bool isInbounds(int x, int y);
00051     void drawCell(SDL_Renderer* renderer, const Cell& cell);
00052     void calculateFlowField();
00053     void calculateDistances();
00054     void calculateFlowDirections();
00055
00056     std::vector<Cell> cells;
00057     const int cellCountX, cellCountY;
00058
00059     SDL_Texture* textureCellWall;
00060     SDL_Texture* textureCellTarget;
00061     SDL_Texture* textureCellSpawner;
00062     SDL_Texture* textureCellEmpty;
00063     SDL_Texture* textureCellArrowUp;
00064     SDL_Texture* textureCellArrowRight;
00065     SDL_Texture* textureCellArrowDown;
00066     SDL_Texture* textureCellArrowLeft;
00067 };

```

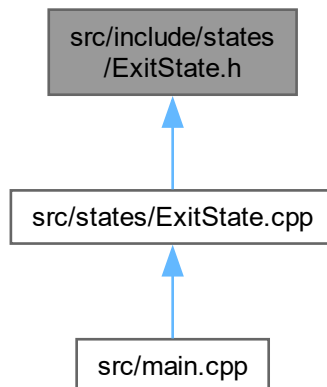
## 5.7 src/include/states/ExitState.h File Reference

```
#include "GameState.h"
```

Include dependency graph for ExitState.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [ExitState](#)

## 5.8 ExitState.h

[Go to the documentation of this file.](#)

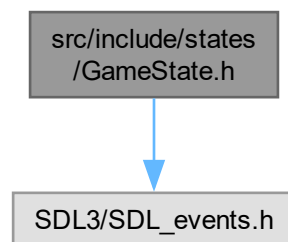
```

00001 #include "GameState.h"
00002
00003 #ifndef EXITSTATE_H
00004 #define EXITSTATE_H
00005 class ExitState : public GameState {
00006 public:
00007     //Static accessor
00008     static ExitState* get();
00009
00010     //Transitions
00011     bool enter() override;
00012     bool exit() override;
00013
00014     //Main loop functions
00015     void handleEvent(SDL_Event& e) override;
00016     void update() override;
00017     void render() override;
00018
00019 private:
00020     //Static instance
00021     static ExitState sExitState;
00022
00023     //Private constructor
00024     ExitState();
00025 };
00026 #endif

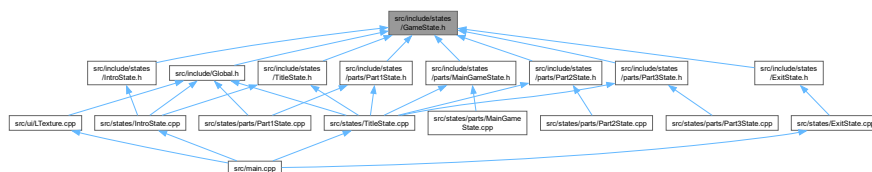
```

## 5.9 src/include/states/GameState.h File Reference

#include <SDL3/SDL\_events.h>  
 Include dependency graph for GameState.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [GameState](#)

## 5.10 GameState.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <SDL3/SDL_events.h>
00003 class GameState {
00004 public:
00005     //State transitions
00006     virtual bool enter() = 0;
00007     virtual bool exit() = 0;
00008
00009     //Main loop functions
00010     virtual void handleEvent(SDL_Event& e) = 0;
00011     virtual void update() = 0;
00012     virtual void render() = 0;
00013
00014     //Make sure to call child destructor
00015     virtual ~GameState() = default;
00016 };

```

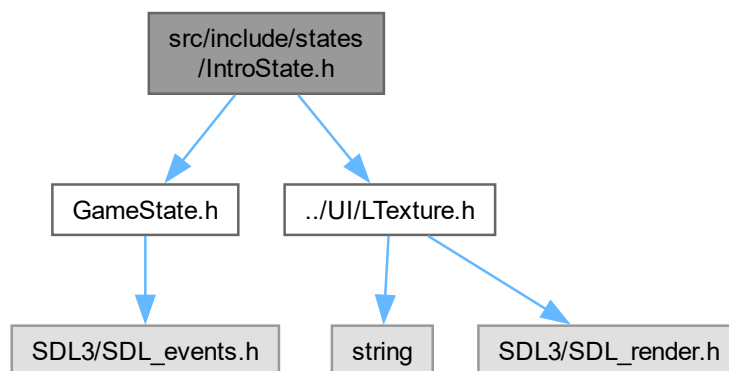
## 5.11 src/include/states/IntroState.h File Reference

```

#include "GameState.h"
#include "../UI/LTexture.h"

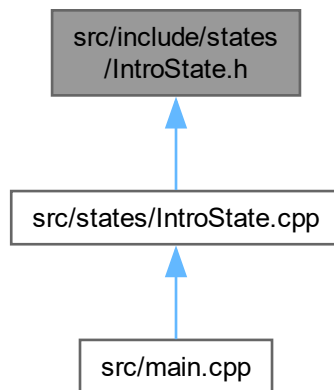
```

Include dependency graph for IntroState.h:





This graph shows which files directly or indirectly include this file:



## Classes

- class [IntroState](#)

## Macros

- `#define` [INTROSTATE\\_H](#)

## 5.11.1 Macro Definition Documentation

### 5.11.1.1 INTROSTATE\_H

```
#define INTROSTATE_H
```

## 5.12 IntroState.h

[Go to the documentation of this file.](#)

```

00001 #include "GameState.h"
00002 #include "../UI/LTexture.h"
00003
00004 #ifndef INTROSTATE_H
00005 #define INTROSTATE_H
00006 class IntroState : public GameState {
00007 public:
00008     //Static accessor
00009     static IntroState* get();
00010
00011     //Transitions
00012     bool enter() override;
00013     bool exit() override;
00014
00015     //Main loop functions
00016     void handleEvent(SDL_Event& e) override;
  
```

```

00017     void update() override;
00018     void render() override;
00019
00020 private:
00021     //Static instance
00022     static IntroState sIntroState;
00023
00024     //Private constructor
00025     IntroState();
00026
00027     //Intro background
00028     LTexture mBackgroundTexture;
00029
00030     //Intro message
00031     LTexture mMessageTexture;
00032 };
00033 #endif

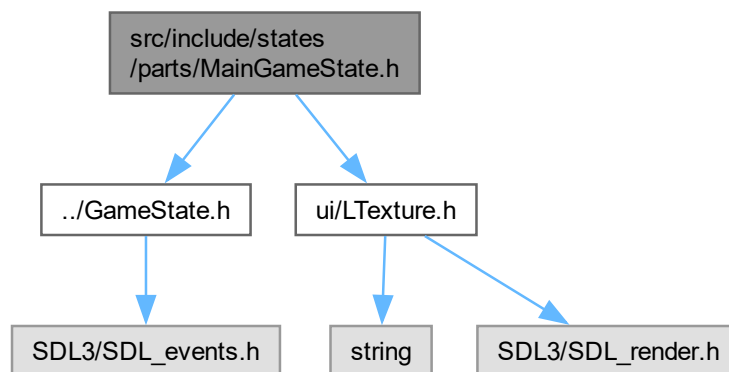
```

### 5.13 src/include/states/parts/MainGameState.h File Reference

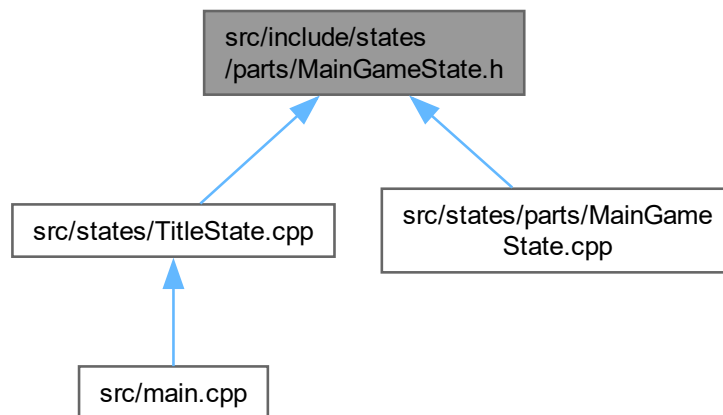
```
#include "../GameState.h"
```

```
#include <ui/LTexture.h>
```

Include dependency graph for MainGameState.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [MainGameState](#)

## 5.14 MainGameState.h

[Go to the documentation of this file.](#)

```

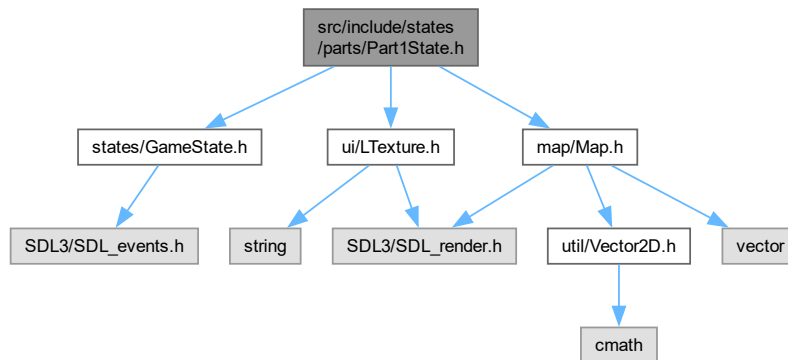
00001 #ifndef MAINGAMESTATE_H
00002 #define MAINGAMESTATE_H
00003
00004 #include "../GameState.h"
00005 #include <ui/LTexture.h>
00006
00007 class MainGameState : public GameState {
00008 public:
00009     //Static accessor
00010     static MainGameState* get();
00011
00012     //Transitions
00013     bool enter() override;
00014     bool exit() override;
00015
00016     //Main loop functions
00017     void handleEvent(SDL_Event& e) override;
00018     void update() override;
00019     void render() override;
00020
00021 private:
00022     //Static instance
00023     static MainGameState sMainGameState;
00024
00025     //Private constructor
00026     MainGameState();
00027
00028     //Intro background
00029     LTexture mBackgroundTexture;
00030
00031     //Intro message
00032     LTexture mMessageTexture;
00033 };
00034 #endif
  
```

## 5.15 src/include/states/parts/Part1State.h File Reference

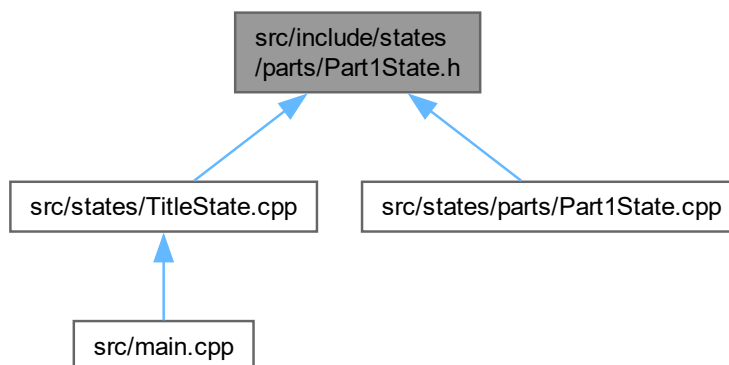
The header file of part 1's ([Map](#)) driver.

```
#include <states/GameState.h>
#include <ui/LTexture.h>
#include <map/Map.h>
```

Include dependency graph for Part1State.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Part1State](#)

### 5.15.1 Detailed Description

The header file of part 1's ([Map](#)) driver.

#### Author

Nathan Grenier

#### Date

2025-02-15

## 5.16 Part1State.h

[Go to the documentation of this file.](#)

```

00001
00007 #pragma once
00008
00009 #include <states/GameState.h>
00010 #include <ui/LTexture.h>
00011 #include <map/Map.h>
00012
00013 class Part1State : public GameState {
00014 public:
00015     //Static accessor
00016     static Part1State* get();
00017
00018     //Transitions
00019     bool enter() override;
00020     bool exit() override;
00021
00022     //Main loop functions
00023     void handleEvent(SDL_Event& e) override;
00024     void update() override;
00025     void render() override;
00026
00027 private:
00028     //Static instance
00029     static Part1State sPart1State;
00030
00031     //Private constructor
00032     Part1State();
00033
00034     //Intro background
00035     LTexture mBackgroundTexture;
00036
00037     //Intro message
00038     LTexture mMessageTexture;
00039
00040     int mouseDownStatus = 0;
00041     int keyDownStatus = 0;
00042
00046     Map* map = nullptr;
00047 };

```

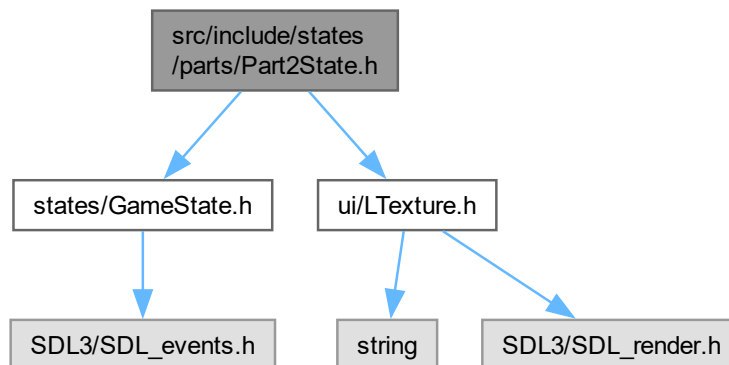
## 5.17 src/include/states/parts/Part2State.h File Reference

```

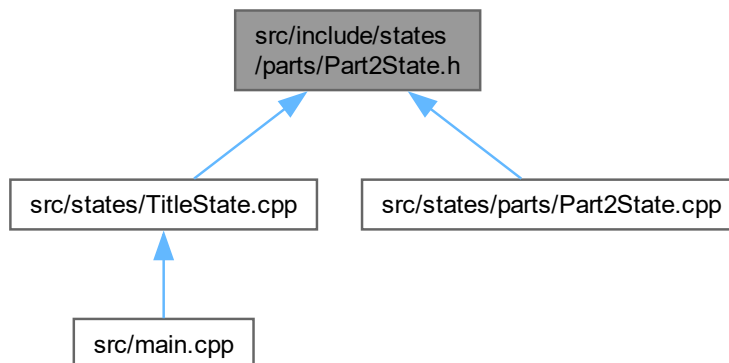
#include <states/GameState.h>
#include <ui/LTexture.h>

```

Include dependency graph for Part2State.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Part2State](#)

## 5.18 Part2State.h

[Go to the documentation of this file.](#)

```

00001 #ifndef PART2STATE_H
00002 #define PART2STATE_H
00003
00004 #include <states/GameState.h>
00005 #include <ui/LTexture.h>

```

```

00006
00007 class Part2State : public GameState {
00008 public:
00009     //Static accessor
00010     static Part2State* get();
00011
00012     //Transitions
00013     bool enter() override;
00014     bool exit() override;
00015
00016     //Main loop functions
00017     void handleEvent(SDL_Event& e) override;
00018     void update() override;
00019     void render() override;
00020
00021 private:
00022     //Static instance
00023     static Part2State sPart2State;
00024
00025     //Private constructor
00026     Part2State();
00027
00028     //Intro background
00029     LTexture mBackgroundTexture;
00030
00031     //Intro message
00032     LTexture mMessageTexture;
00033 };
00034 #endif

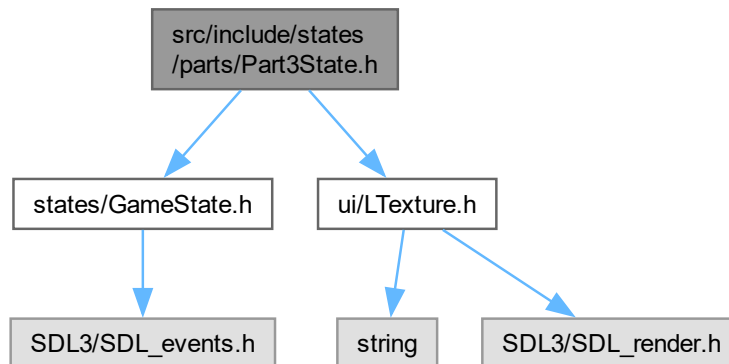
```

## 5.19 src/include/states/parts/Part3State.h File Reference

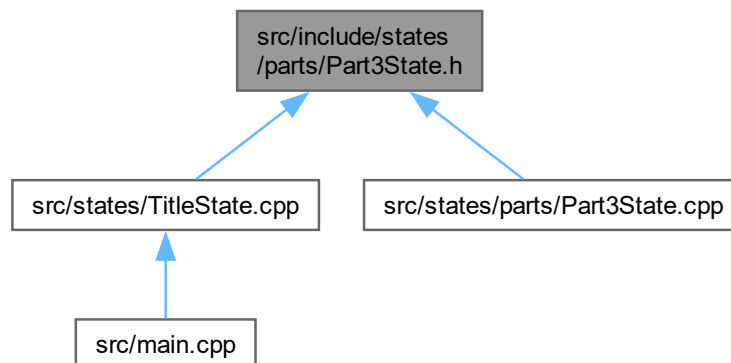
```
#include <states/GameState.h>
```

```
#include <ui/LTexture.h>
```

Include dependency graph for Part3State.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Part3State](#)

## 5.20 Part3State.h

[Go to the documentation of this file.](#)

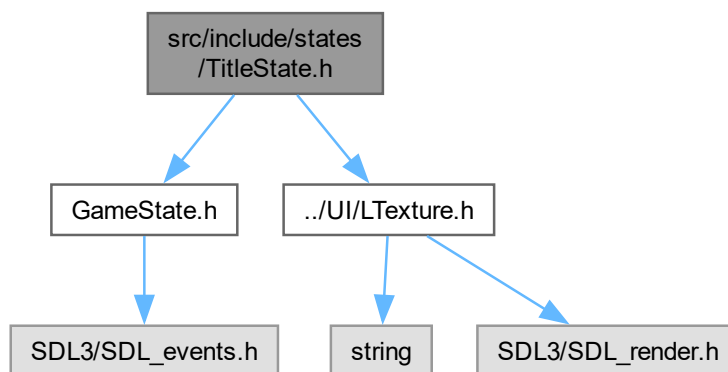
```

00001 #ifndef PART3STATE_H
00002 #define PART3STATE_H
00003
00004 #include <states/GameState.h>
00005 #include <ui/LTexture.h>
00006
00007 class Part3State : public GameState {
00008 public:
00009     //Static accessor
00010     static Part3State* get();
00011
00012     //Transitions
00013     bool enter() override;
00014     bool exit() override;
00015
00016     //Main loop functions
00017     void handleEvent(SDL_Event& e) override;
00018     void update() override;
00019     void render() override;
00020
00021 private:
00022     //Static instance
00023     static Part3State sPart3State;
00024
00025     //Private constructor
00026     Part3State();
00027
00028     //Intro background
00029     LTexture mBackgroundTexture;
00030
00031     //Intro message
00032     LTexture mMessageTexture;
00033 };
00034 #endif
  
```

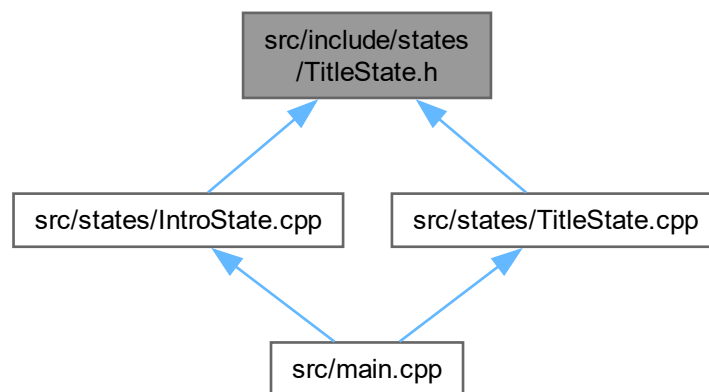


## 5.21 src/include/states/TitleState.h File Reference

```
#include "GameState.h"  
#include "../UI/LTexture.h"  
Include dependency graph for TitleState.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [TitleState](#)

## 5.22 TitleState.h

[Go to the documentation of this file.](#)

```

00001 #include "GameState.h"
00002 #include "../UI/LTexture.h"
00003
00004 #ifndef TITLESTATE_H
00005 #define TITLESTATE_H
00006 class TitleState : public GameState {
00007 public:
00008     //Static accessor
00009     static TitleState* get();
00010
00011     //Transitions
00012     bool enter() override;
00013     bool exit() override;
00014
00015     //Main loop functions
00016     void handleEvent(SDL_Event& e) override;
00017     void update() override;
00018     void render() override;
00019
00020 private:
00021     //Static instance
00022     static TitleState sTitleState;
00023
00024     //Private constructor
00025     TitleState();
00026
00027     //Intro background
00028     LTexture mBackgroundTexture;
00029
00030     //Intro message
00031     LTexture mMessageTexture;
00032 };
00033 #endif

```

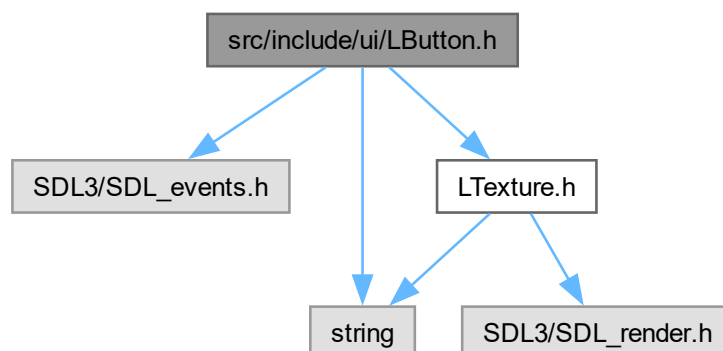
## 5.23 src/include/ui/LButton.h File Reference

```

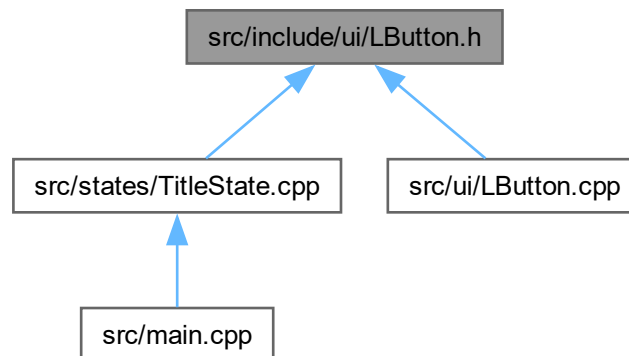
#include <SDL3/SDL_events.h>
#include <string>
#include "LTexture.h"

```

Include dependency graph for LButton.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [LButton](#)

## 5.24 LButton.h

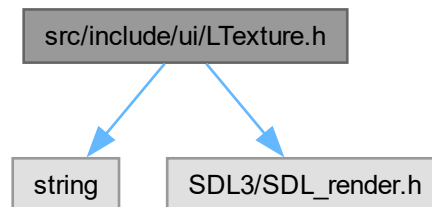
[Go to the documentation of this file.](#)

```

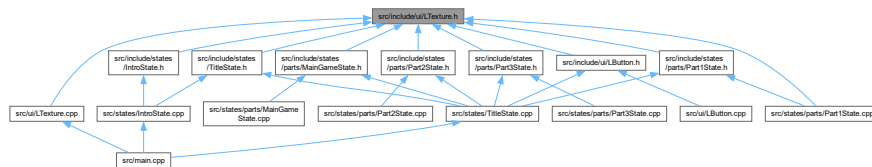
00001 #pragma once
00002 #include <SDL3/SDL_events.h>
00003 #include <string>
00004 #include "LTexture.h"
00005 class LButton {
00006 public:
00007     //Button dimensions
00008     static constexpr int kButtonWidth = 300;
00009     static constexpr int kButtonHeight = 50;
00010
00011     //Initializes internal variables
00012     LButton();
00013
00014     //Sets top left position
00015     void setPosition(float x, float y);
00016
00017     //Handles mouse event
00018     void handleEvent(SDL_Event* e);
00019
00020     bool setText(const std::string& text, SDL_Color textColor);
00021
00022     //Shows button sprite
00023     void render();
00024
00025     bool isClicked() const;
00026
00027 private:
00028     enum eButtonSprite {
00029         eButtonSpriteMouseOut = 0,
00030         eButtonSpriteMouseOverMotion = 1,
00031         eButtonSpriteMouseDown = 2,
00032         eButtonSpriteMouseUp = 3
00033     };
00034
00035     //Top left position
00036     SDL_FPoint mPosition;
00037
00038     //Currently used global sprite
00039     eButtonSprite mCurrentSprite;
00040
00041     LTexture gButtonSpriteTexture;
00042 };
  
```

## 5.25 src/include/ui/LTexture.h File Reference

```
#include <string>
#include <SDL3/SDL_render.h>
Include dependency graph for LTexture.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [LTexture](#)

## 5.26 LTexture.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <SDL3/SDL_render.h>
00005 class LTexture {
00006 public:
00007     //Symbolic constant
00008     static constexpr float kOriginalSize = -1.f;
00009
00010     //Initializes texture variables
00011     LTexture();
00012
00013     //Cleans up texture variables
00014     ~LTexture();
00015
00016     //Loads texture from disk
00017     bool loadFromFile(std::string path);
00018
00019     bool loadFromRenderedText(std::string textureText, SDL_Color textColor);
00020

```

```

00021     //Cleans up texture
00022     void destroy();
00023
00024     //Sets color modulation
00025     void setColor(UInt8 r, UInt8 g, UInt8 b);
00026
00027     //Sets opacity
00028     void setAlpha(UInt8 alpha);
00029
00030     //Sets blend mode
00031     void setBlending(SDL_BlendMode blendMode);
00032
00033     //Draws texture
00034     void render(float x, float y, SDL_FRect* clip = nullptr, float width = kOriginalSize, float height
= kOriginalSize, double degrees = 0.0, SDL_FPoint* center = nullptr, SDL_FlipMode flipMode =
SDL_FLIP_NONE);
00035
00036     //Gets texture dimensions
00037     int getWidth();
00038     int getHeight();
00039
00040 private:
00041     //Contains texture data
00042     SDL_Texture* mTexture;
00043
00044     //Texture dimensions
00045     int mWidth;
00046     int mHeight;
00047 };

```

## 5.27 src/include/util/TextureLoader.h File Reference

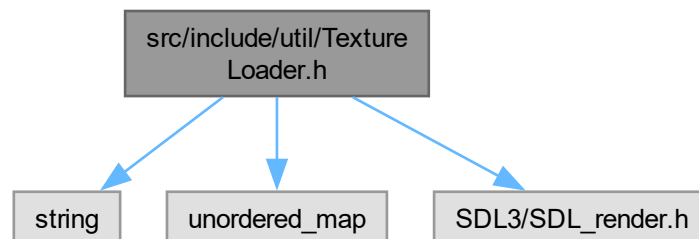
Load and store textures.

```

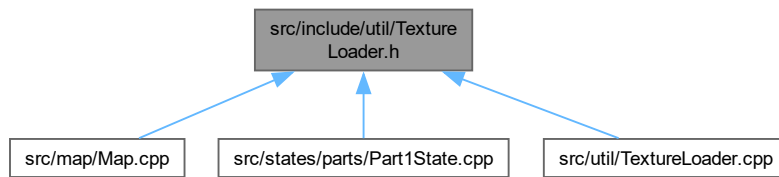
#include <string>
#include <unordered_map>
#include <SDL3/SDL_render.h>

```

Include dependency graph for TextureLoader.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [TextureLoader](#)

### 5.27.1 Detailed Description

Load and store textures.

#### Author

Nathan Grenier

#### Date

2025-02-15

## 5.28 TextureLoader.h

[Go to the documentation of this file.](#)

```

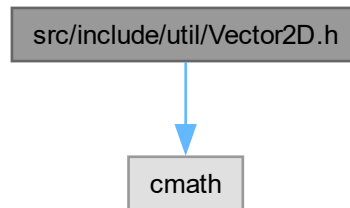
00001
00007 #pragma once
00008
00009 #include <string>
00010 #include <unordered_map>
00011 #include <SDL3/SDL_render.h>
00012
00013 class TextureLoader {
00014 public:
00015     static SDL_Texture* loadTexture(SDL_Renderer* renderer, std::string filename);
00016     static void deallocateTextures();
00017
00018 private:
00019     static std::unordered_map<std::string, SDL_Texture*> loadedTextures;
00020     static inline const std::string TEXTURE_PATH = "assets/";
00021 };
  
```

## 5.29 src/include/util/Vector2D.h File Reference

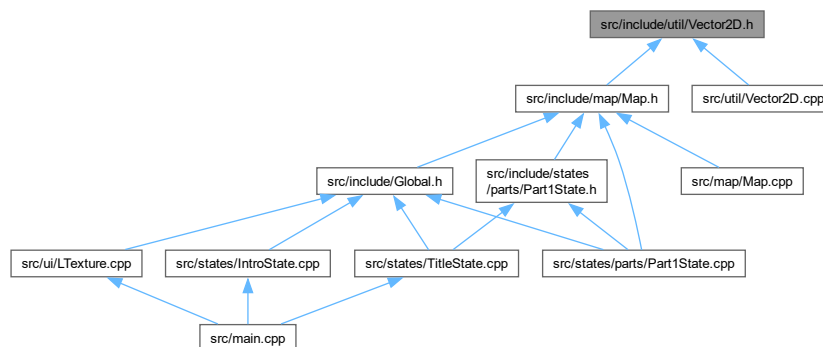
Representation of a 2D vector.

```
#include <cmath>
```

Include dependency graph for Vector2D.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Vector2D](#)

### 5.29.1 Detailed Description

Representation of a 2D vector.

## 5.30 Vector2D.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include <cmath>
00009
00010 class Vector2D {
00011 public:
00012     Vector2D(float setX, float setY) : x(setX), y(setY) {}
00013     Vector2D(const Vector2D& other) : x(other.x), y(other.y) {}
00014     Vector2D(float angleRad) : x(cos(angleRad)), y(sin(angleRad)) {}
00015     Vector2D() : x(0.0f), y(0.0f) {}
00016
00017     float angle() { return atan2(y, x); }
00018
00019     float magnitude() { return sqrt(x * x + y * y); }
00020     Vector2D normalize();
00021     Vector2D getNegativeReciprocal() { return Vector2D(-y, x); }
00022
00023     float dot(const Vector2D& other) { return x * other.x + y * other.y; }
00024     float cross(const Vector2D& other) { return x * other.y - y * other.x; }
00025     float angleBetween(const Vector2D& other) { return atan2(cross(other), dot(other)); }
00026
00027     Vector2D operator+(const float amount) { return Vector2D(x + amount, y + amount); }
00028     Vector2D operator-(const float amount) { return Vector2D(x - amount, y - amount); }
00029     Vector2D operator*(const float amount) { return Vector2D(x * amount, y * amount); }
00030     Vector2D operator/(const float amount) { return Vector2D(x / amount, y / amount); }
00031
00032     Vector2D operator+(const Vector2D& other) { return Vector2D(x + other.x, y + other.y); }
00033     Vector2D operator-(const Vector2D& other) { return Vector2D(x - other.x, y - other.y); }
00034     Vector2D operator*(const Vector2D& other) { return Vector2D(x * other.x, y * other.y); }
00035     Vector2D operator/(const Vector2D& other) { return Vector2D(x / other.x, y / other.y); }
00036
00037     Vector2D& operator+=(const float amount) { x += amount; y += amount; return *this; }
00038     Vector2D& operator-=(const float amount) { x -= amount; y -= amount; return *this; }
00039     Vector2D& operator*=(const float amount) { x *= amount; y *= amount; return *this; }
00040     Vector2D& operator/=(const float amount) { x /= amount; y /= amount; return *this; }
00041
00042     Vector2D& operator+=(const Vector2D& other) { x += other.x; y += other.y; return *this; }
00043     Vector2D& operator-=(const Vector2D& other) { x -= other.x; y -= other.y; return *this; }
00044     Vector2D& operator*=(const Vector2D& other) { x *= other.x; y *= other.y; return *this; }
00045     Vector2D& operator/=(const Vector2D& other) { x /= other.x; y /= other.y; return *this; }
00046
00047     float x, y;
00048 };

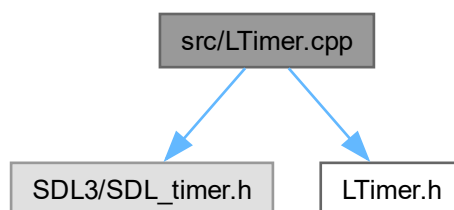
```

## 5.31 src/LTimer.cpp File Reference

```
#include <SDL3/SDL_timer.h>
```

```
#include <LTimer.h>
```

Include dependency graph for LTimer.cpp:

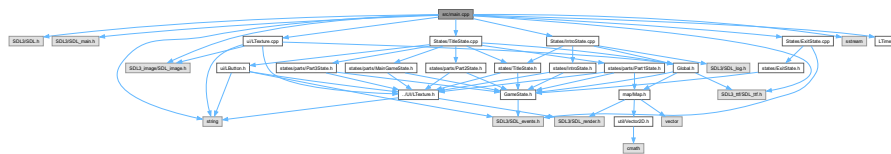




## 5.32 src/main.cpp File Reference

```
#include <SDL3/SDL.h>
#include <SDL3/SDL_main.h>
#include <SDL3_image/SDL_image.h>
#include <SDL3_ttf/SDL_ttf.h>
#include <string>
#include <sstream>
#include "States/TitleState.cpp"
#include "States/IntroState.cpp"
#include "States/ExitState.cpp"
#include "ui/LTexture.cpp"
#include <LTimer.h>
```

Include dependency graph for main.cpp:



### Functions

- void [setNextState](#) (GameState \*newState)
- void [changeState](#) ()
- bool [init](#) ()
- bool [loadMedia](#) ()
- void [close](#) ()
- int [main](#) (int argc, char \*args[])

### Variables

- constexpr int [kScreenFps](#) { 60 }
- SDL\_Window \* [gWindow](#) { nullptr }
- SDL\_Renderer \* [gRenderer](#) = nullptr
- TTF\_Font \* [gFont](#) = nullptr
- LTexture [gFpsTexture](#)
- GameState \* [gCurrentState](#) { nullptr }
- GameState \* [gNextState](#) { nullptr }

### 5.32.1 Function Documentation

#### 5.32.1.1 changeState()

```
void changeState ()
```

Here is the caller graph for this function:



#### 5.32.1.2 `close()`

```
void close ()
```

Here is the caller graph for this function:



#### 5.32.1.3 `init()`

```
bool init ()
```

Here is the caller graph for this function:



#### 5.32.1.4 loadMedia()

```
bool loadMedia ()
```

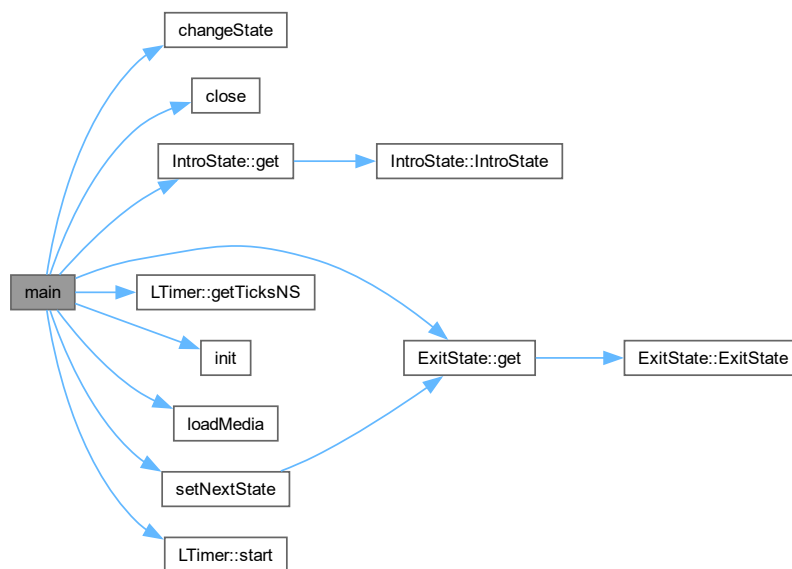
Here is the caller graph for this function:



#### 5.32.1.5 main()

```
int main (  
    int argc,  
    char * args[])
```

Here is the call graph for this function:



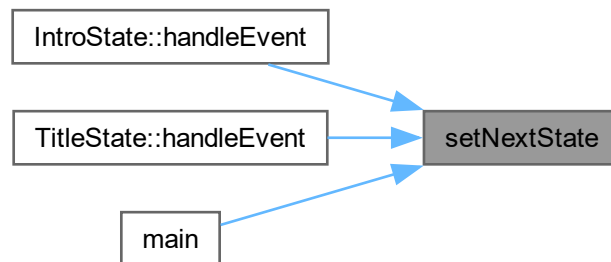
### 5.32.1.6 setNextState()

```
void setNextState (
    GameState * newState)
```

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.32.2 Variable Documentation

### 5.32.2.1 gCurrentState

```
GameState* gCurrentState { nullptr }
```

### 5.32.2.2 gFont

```
TTF_Font* gFont = nullptr
```

### 5.32.2.3 gFpsTexture

```
LTexture gFpsTexture
```

### 5.32.2.4 gNextState

```
GameState* gNextState { nullptr }
```

### 5.32.2.5 gRenderer

```
SDL_Renderer* gRenderer = nullptr
```

### 5.32.2.6 gWindow

```
SDL_Window* gWindow { nullptr }
```

### 5.32.2.7 kScreenFps

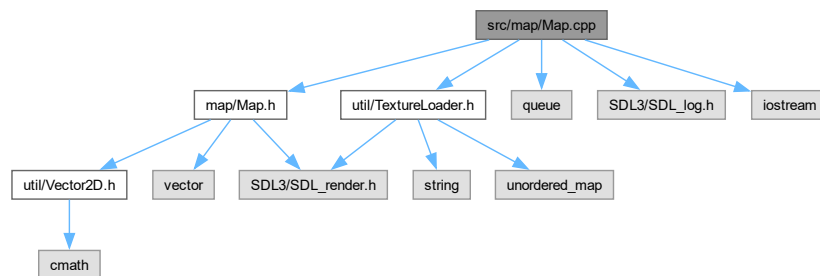
```
int kScreenFps { 60 } [constexpr]
```

## 5.33 src/map/Map.cpp File Reference

Implementation of the [Map](#) class.

```
#include <map/Map.h>
#include <util/TextureLoader.h>
#include <queue>
#include <SDL3/SDL_log.h>
#include <iostream>
```

Include dependency graph for Map.cpp:

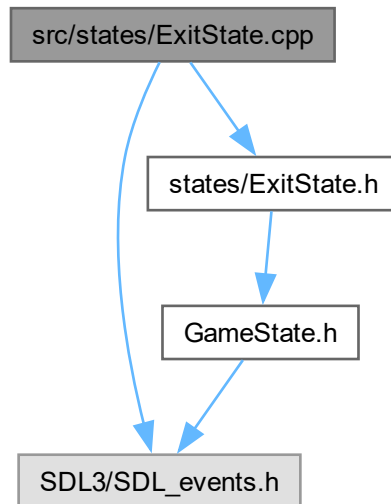


### 5.33.1 Detailed Description

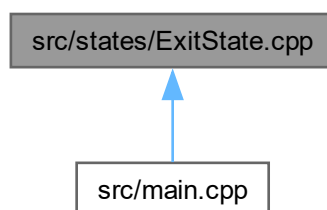
Implementation of the [Map](#) class.

### 5.34 src/states/ExitState.cpp File Reference

```
#include <SDL3/SDL_events.h>
#include <states/ExitState.h>
Include dependency graph for ExitState.cpp:
```



This graph shows which files directly or indirectly include this file:



### 5.35 ExitState.cpp

[Go to the documentation of this file.](#)

```
00001 #include <SDL3/SDL_events.h>
00002 #include <states/ExitState.h>
00003
00004 ExitState ExitState::sExitState;
00005
```

```

00006 //Hollow exit state
00007 ExitState* ExitState::get() {
00008     return &sExitState;
00009 }
00010
00011 bool ExitState::enter() {
00012     return true;
00013 }
00014
00015 bool ExitState::exit() {
00016     return true;
00017 }
00018
00019 void ExitState::handleEvent(SDL_Event& e) {
00020
00021 }
00022
00023 void ExitState::update() {
00024
00025 }
00026
00027 void ExitState::render() {
00028
00029 }
00030
00031 ExitState::ExitState() {
00032
00033 }

```

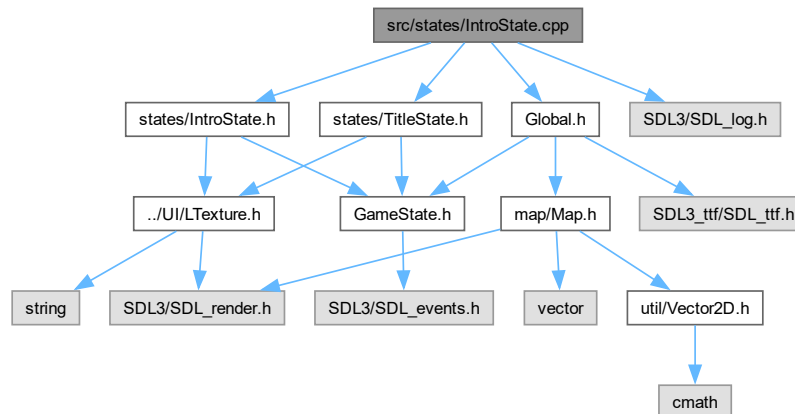
## 5.36 src/states/IntroState.cpp File Reference

```

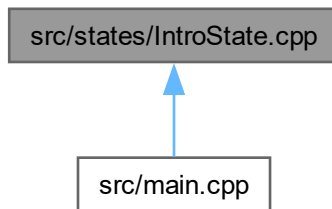
#include <states/IntroState.h>
#include <SDL3/SDL_log.h>
#include <states/TitleState.h>
#include <Global.h>

```

Include dependency graph for IntroState.cpp:



This graph shows which files directly or indirectly include this file:



## 5.37 IntroState.cpp

[Go to the documentation of this file.](#)

```

00001 #include <states/IntroState.h>
00002 #include <SDL3/SDL_log.h>
00003 #include <states/TitleState.h>
00004 #include <Global.h>
00005
00006 IntroState IntroState::sIntroState;
00007
00008 //IntroState Implementation
00009 IntroState* IntroState::get() {
00010     //Get static instance
00011     return &sIntroState;
00012 }
00013
00014 bool IntroState::enter() {
00015     //Loading success flag
00016     bool success = true;
00017
00018     //Load text
00019     SDL_Color textColor{ 0x00, 0x00, 0x00, 0xFF };
00020     if (success &= mMessageTexture.loadFromRenderedText("The NullTerminators Present...", textColor);
00021 !success)
00022     {
00023         SDL_Log("Failed to render intro text!\n");
00024         success = false;
00025     }
00026     return success;
00027 }
00028
00029 bool IntroState::exit() {
00030     //Free background and text
00031     mBackgroundTexture.destroy();
00032     mMessageTexture.destroy();
00033     return true;
00034 }
00035
00036
00037 void IntroState::handleEvent(SDL_Event& e) {
00038     //If the user pressed enter
00039     if ((e.type == SDL_EVENT_MOUSE_BUTTON_DOWN))
00040     {
00041         //Move onto title state
00042         setNextState(TitleState::get());
00043     }
00044 }
00045
00046 void IntroState::update() {
00047 }
00048
00049
00050 void IntroState::render() {
00051     //Show the background
00052     mBackgroundTexture.render(0, 0);
  
```



```

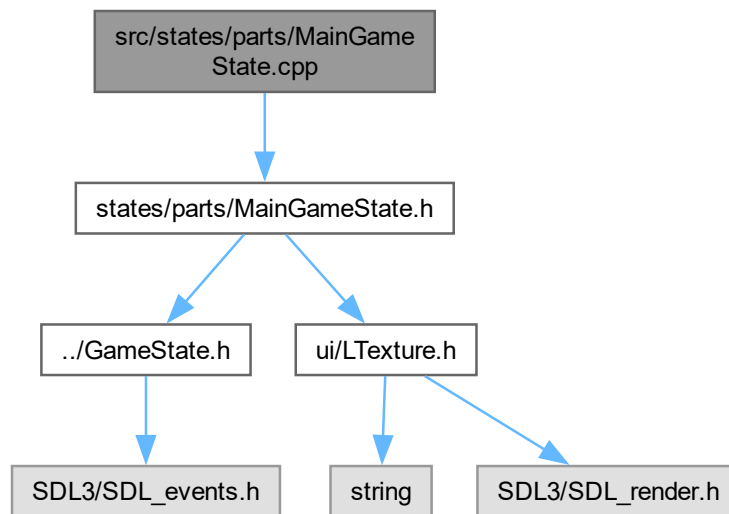
00053
00054     //Show the message
00055     mMessageTexture.render((Global::kScreenWidth - mMessageTexture.getWidth()) / 2.f,
00056                             (Global::kScreenHeight - mMessageTexture.getHeight()) / 2.f);
00056 }
00057
00058 IntroState::IntroState() {}

```

## 5.38 src/states/parts/MainGameState.cpp File Reference

```
#include <states/parts/MainGameState.h>
```

Include dependency graph for MainGameState.cpp:



## 5.39 src/states/parts/Part1State.cpp File Reference

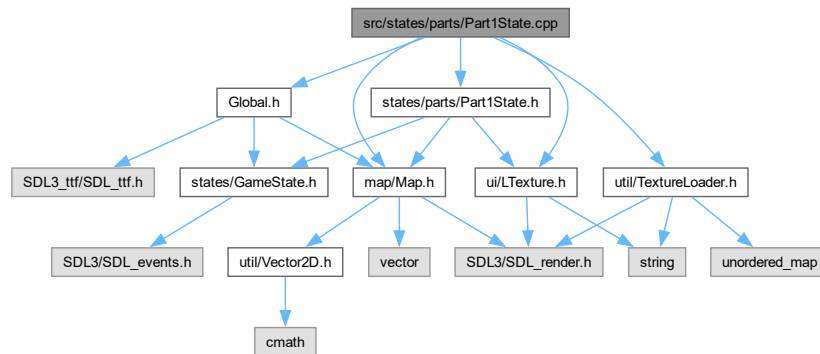
The drive file for part 1 of assignment 1 ([Map](#))

```

#include <states/parts/Part1State.h>
#include <Global.h>
#include <util/TextureLoader.h>
#include <map/Map.h>
#include <ui/LTexture.h>

```

Include dependency graph for Part1State.cpp:



### 5.39.1 Detailed Description

The drive file for part 1 of assignment 1 ([Map](#))

Author

Nathan Grenier

Date

2025-02-15

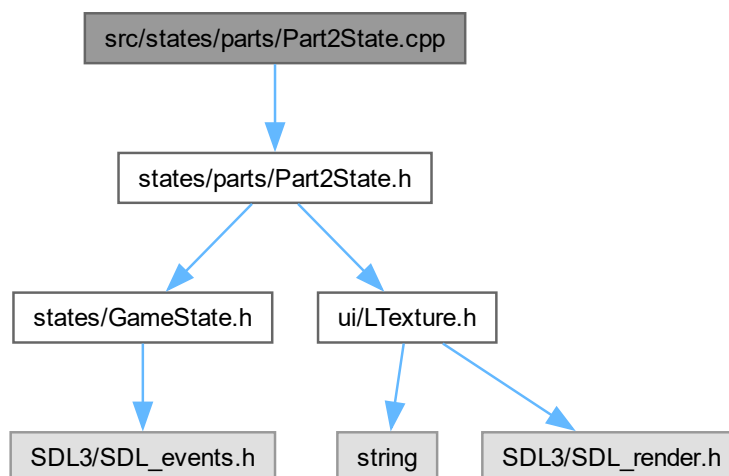
Controls:

- Left Click: Place a wall.
- Right Click: Remove a wall.
- Shift + Left Click: Set the target cell.
- Shift + Right Click: Set the spawner cell.

## 5.40 src/states/parts/Part2State.cpp File Reference

```
#include <states/parts/Part2State.h>
```

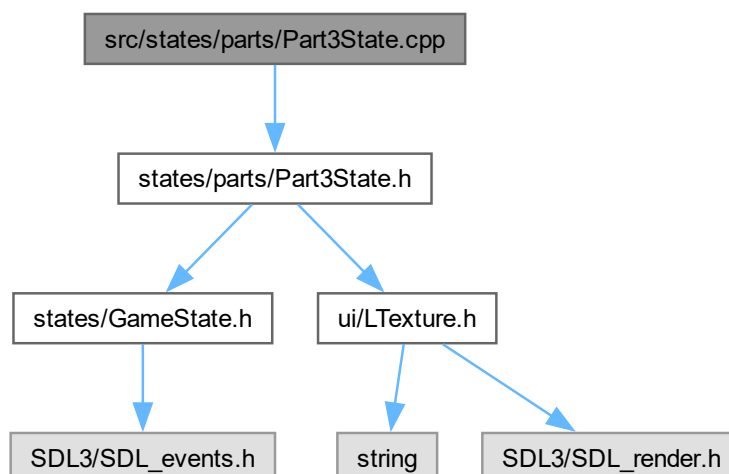
Include dependency graph for Part2State.cpp:



## 5.41 src/states/parts/Part3State.cpp File Reference

```
#include <states/parts/Part3State.h>
```

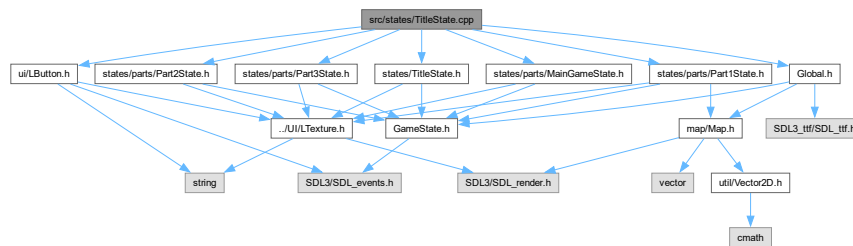
Include dependency graph for Part3State.cpp:



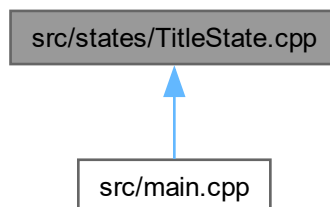
## 5.42 src/states/TitleState.cpp File Reference

```
#include <states/TitleState.h>
#include <ui/LButton.h>
#include <states/parts/MainGameState.h>
#include <states/parts/Part1State.h>
#include <states/parts/Part2State.h>
#include <states/parts/Part3State.h>
#include <Global.h>
```

Include dependency graph for TitleState.cpp:



This graph shows which files directly or indirectly include this file:



### Variables

- constexpr int `kButtonCount` = 4
- `LButton` `buttons[kButtonCount]`

### 5.42.1 Variable Documentation

#### 5.42.1.1 buttons

```
LButton buttons[kButtonCount]
```

## 5.42.1.2 kButtonCount

```
int kButtonCount = 4 [constexpr]
```

## 5.43 TitleState.cpp

[Go to the documentation of this file.](#)

```
00001 #include <states/TitleState.h>
00002 #include <ui/LButton.h>
00003 #include <states/parts/MainGameState.h>
00004 #include <states/parts/Part1State.h>
00005 #include <states/parts/Part2State.h>
00006 #include <states/parts/Part3State.h>
00007 #include <Global.h>
00008
00009 TitleState TitleState::sTitleState;
00010
00011 //Place buttons
00012 constexpr int kButtonCount = 4;
00013 LButton buttons[kButtonCount];
00014
00015 //TitleState Implementation
00016 TitleState* TitleState::get() {
00017     //Get static instance
00018     return &sTitleState;
00019 }
00020
00021 bool TitleState::enter() {
00022     bool success = true;
00023
00024     SDL_Color textColor{ 0x00, 0x00, 0x00, 0xFF };
00025
00026     if (!(success &= mMessageTexture.loadFromRenderedText("Tower Defense - The Game", textColor)))
00027     {
00028         printf("Failed to render title text!\n");
00029     }
00030
00031     // Assign text to buttons
00032     const char* buttonLabels[kButtonCount] = {
00033         "Load Main Game",
00034         "Load Part 1",
00035         "Load Part 2",
00036         "Load Part 3"
00037     };
00038
00039     for (int i = 0; i < kButtonCount; ++i)
00040     {
00041         if (!buttons[i].setText(buttonLabels[i], textColor))
00042         {
00043             printf("Failed to set button text: %s\n", buttonLabels[i]);
00044             success = false;
00045         }
00046     }
00047
00048     return success;
00049 }
00050
00051
00052 bool TitleState::exit() {
00053     //Free background and text
00054     mBackgroundTexture.destroy();
00055     mMessageTexture.destroy();
00056
00057     return true;
00058 }
00059
00060 void TitleState::handleEvent(SDL_Event& e) {
00061     //Handle button events
00062     for (int i = 0; i < kButtonCount; ++i)
00063     {
00064         buttons[i].handleEvent(&e);
00065
00066         // Check if any of the buttons were clicked and set the next state accordingly
00067         if (e.type == SDL_EVENT_MOUSE_BUTTON_DOWN && e.button.button == SDL_BUTTON_LEFT)
00068         {
00069             if (buttons[i].isClicked())
00070             {
00071                 // Transition to the corresponding part state
00072                 switch (i)
```

```

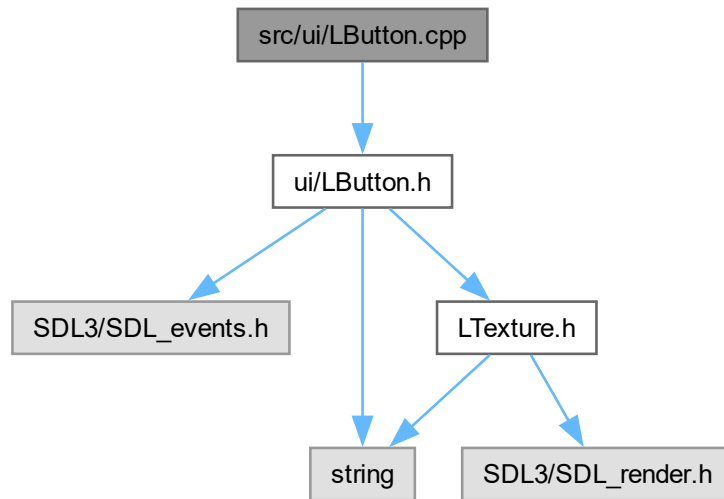
00073         {
00074             case 0: // "Load Main Game" - Can stay in TitleState or go to main game
00075                 setNextState(MainGameState::get()); // Assuming you have a MainGameState
00076                 break;
00077
00078             case 1: // "Load Part 1"
00079                 setNextState(Part1State::get());
00080                 break;
00081
00082             case 2: // "Load Part 2"
00083                 setNextState(Part2State::get());
00084                 break;
00085
00086             case 3: // "Load Part 3"
00087                 setNextState(Part3State::get());
00088                 break;
00089             }
00090         }
00091     }
00092 }
00093 }
00094
00095 void TitleState::update() {
00096     //Fill the background
00097     SDL_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);
00098     SDL_RenderClear(gRenderer);
00099 }
00100
00101 void TitleState::render() {
00102     // Define vertical spacing
00103     constexpr int buttonSpacing = 20; // Space between buttons
00104     constexpr int startY = (Global::kScreenHeight - ((LButton::kButtonHeight * kButtonCount) +
00105         (buttonSpacing * (kButtonCount - 1)))) / 2;
00106
00107     // Set button positions
00108     for (int i = 0; i < kButtonCount; ++i)
00109     {
00110         buttons[i].setPosition((Global::kScreenWidth - LButton::kButtonWidth) / 2, startY + i *
00111             (LButton::kButtonHeight + buttonSpacing));
00112     }
00113
00114     // Show the background
00115     mBackgroundTexture.render(0, 0);
00116
00117     // Show the message
00118     mMessageTexture.render((Global::kScreenWidth - mMessageTexture.getWidth()) / 2.f, 20);
00119
00120     // Render buttons
00121     for (int i = 0; i < kButtonCount; i++)
00122     {
00123         buttons[i].render();
00124     }
00125 }
00126
00127 TitleState::TitleState() {
00128     //No public instantiation
00129 }

```

## 5.44 src/ui/LButton.cpp File Reference

```
#include <ui/LButton.h>
```

Include dependency graph for LButton.cpp:



## 5.45 src/ui/LTexture.cpp File Reference

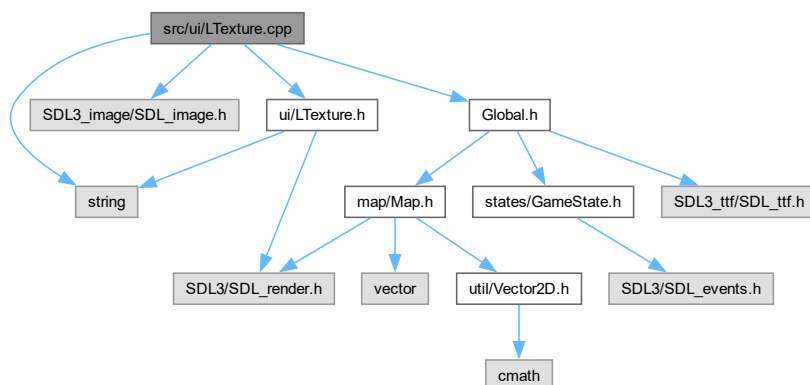
```
#include <string>
```

```
#include <SDL3_image/SDL_image.h>
```

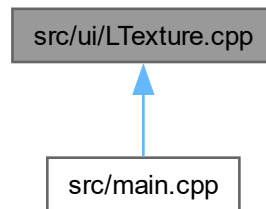
```
#include <ui/LTexture.h>
```

```
#include <Global.h>
```

Include dependency graph for LTexture.cpp:



This graph shows which files directly or indirectly include this file:



## 5.46 LTexture.cpp

[Go to the documentation of this file.](#)

```

00001 #include <string>
00002 #include <SDL3_image/SDL_image.h>
00003 #include <ui/LTexture.h>
00004 #include <Global.h>
00005
00006 //LTexture Implementation
00007 LTexture::LTexture() :
00008     //Initialize texture variables
00009     mTexture{ nullptr },
00010     mWidth{ 0 },
00011     mHeight{ 0 } {
00012 }
00013 }
00014
00015 LTexture::~LTexture() {
00016     //Clean up texture
00017     destroy();
00018 }
00019
00020 bool LTexture::loadFromFile(std::string path) {
00021     //Clean up texture if it already exists
00022     destroy();
00023
00024     //Load surface
00025     if (SDL_Surface* loadedSurface = IMG_Load(path.c_str()); loadedSurface == nullptr)
00026     {
00027         SDL_Log("Unable to load image %s! SDL_image error: %s\n", path.c_str(), SDL_GetError());
00028     } else
00029     {
00030         //Color key image
00031         if (!SDL_SetSurfaceColorKey(loadedSurface, true, SDL_MapSurfaceRGB(loadedSurface, 0x00, 0xFF,
00032 0xFF)))
00033         {
00034             SDL_Log("Unable to color key! SDL error: %s", SDL_GetError());
00035         } else
00036         {
00037             //Create texture from surface
00038             if (mTexture = SDL_CreateTextureFromSurface(gRenderer, loadedSurface); mTexture ==
00039 nullptr)
00040             {
00041                 SDL_Log("Unable to create texture from loaded pixels! SDL error: %s\n",
00042 SDL_GetError());
00043             } else
00044             {
00045                 //Get image dimensions
00046                 mWidth = loadedSurface->w;
00047                 mHeight = loadedSurface->h;
00048             }
00049         }
00050         //Clean up loaded surface
00051         SDL_DestroySurface(loadedSurface);
00052     }
  
```



```

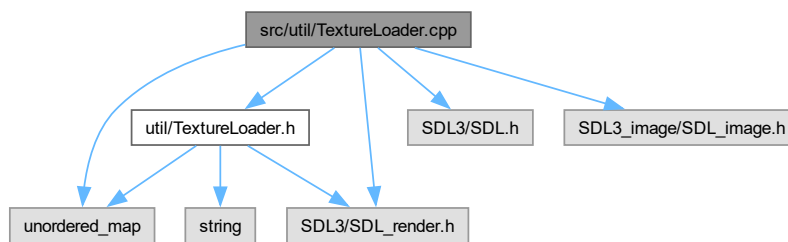
00051
00052     //Return success if texture loaded
00053     return mTexture != nullptr;
00054 }
00055
00056 bool LTexture::loadFromRenderedText(std::string textureText, SDL_Color textColor) {
00057     //Clean up existing texture
00058     destroy();
00059
00060     //Load text surface
00061     if (SDL_Surface* textSurface = TTF_RenderText_Blended(gFont, textureText.c_str(), 0, textColor);
00062         textSurface == nullptr)
00063     {
00064         SDL_Log("Unable to render text surface! SDL_ttf Error: %s\n", SDL_GetError());
00065     } else
00066     {
00067         //Create texture from surface
00068         if (mTexture = SDL_CreateTextureFromSurface(gRenderer, textSurface); mTexture == nullptr)
00069         {
00070             SDL_Log("Unable to create texture from rendered text! SDL Error: %s\n", SDL_GetError());
00071         } else
00072         {
00073             mWidth = textSurface->w;
00074             mHeight = textSurface->h;
00075         }
00076
00077         //Free temp surface
00078         SDL_DestroySurface(textSurface);
00079     }
00080
00081     //Return success if texture loaded
00082     return mTexture != nullptr;
00083 }
00084 void LTexture::destroy() {
00085     //Clean up texture
00086     SDL_DestroyTexture(mTexture);
00087     mTexture = nullptr;
00088     mWidth = 0;
00089     mHeight = 0;
00090 }
00091
00092 void LTexture::setColor(Uint8 r, Uint8 g, Uint8 b) {
00093     SDL_SetTextureColorMod(mTexture, r, g, b);
00094 }
00095
00096 void LTexture::setAlpha(Uint8 alpha) {
00097     SDL_SetTextureAlphaMod(mTexture, alpha);
00098 }
00099
00100 void LTexture::setBlending(SDL_BlendMode blendMode) {
00101     SDL_SetTextureBlendMode(mTexture, blendMode);
00102 }
00103
00104 void LTexture::render(float x, float y, SDL_FRect* clip, float width, float height, double degrees,
00105     SDL_FPoint* center, SDL_FlipMode flipMode) {
00106     //Set texture position
00107     SDL_FRect dstRect = { x, y, static_cast<float>(mWidth), static_cast<float>(mHeight) };
00108
00109     //Default to clip dimensions if clip is given
00110     if (clip != nullptr)
00111     {
00112         dstRect.w = clip->w;
00113         dstRect.h = clip->h;
00114     }
00115
00116     //Resize if new dimensions are given
00117     if (width > 0)
00118     {
00119         dstRect.w = width;
00120     }
00121     if (height > 0)
00122     {
00123         dstRect.h = height;
00124     }
00125
00126     //Render texture
00127     SDL_RenderTextureRotated(gRenderer, mTexture, clip, &dstRect, degrees, center, flipMode);
00128 }
00129 int LTexture::getWidth() {
00130     return mWidth;
00131 }
00132
00133 int LTexture::getHeight() {
00134     return mHeight;
00135 }

```

## 5.47 src/util/TextureLoader.cpp File Reference

Implementation of the Texture Loader Class.

```
#include <util/TextureLoader.h>
#include <SDL3/SDL_render.h>
#include <unordered_map>
#include <SDL3/SDL.h>
#include <SDL3_image/SDL_image.h>
Include dependency graph for TextureLoader.cpp:
```



### Functions

- static `std::unordered_map< std::string, SDL_Texture * > & getLoadedTextures ()`  
*Gets the static map of loaded textures.*

### 5.47.1 Detailed Description

Implementation of the Texture Loader Class.

This file implements a texture loading system that manages SDL textures, including loading from files and memory management. It implements texture caching to prevent loading the same texture multiple times.

### 5.47.2 Function Documentation

#### 5.47.2.1 getLoadedTextures()

```
static std::unordered_map< std::string, SDL_Texture * > & getLoadedTextures () [static]
```

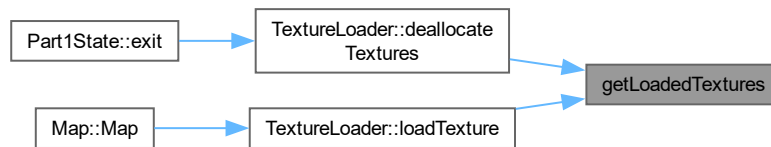
Gets the static map of loaded textures.

This function implements the Singleton pattern for the texture cache, ensuring there's only one instance of the texture map throughout the program's lifetime.

### Returns

Reference to the static map of loaded textures

Here is the caller graph for this function:

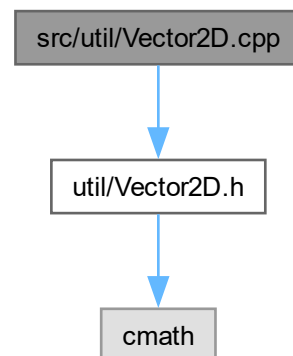


## 5.48 src/util/Vector2D.cpp File Reference

Implementation of a the [Vector2D](#) class.

```
#include <util/Vector2D.h>
```

Include dependency graph for Vector2D.cpp:



### 5.48.1 Detailed Description

Implementation of a the [Vector2D](#) class.

Most of the implementation can be found in [Vector2D.h](#). Most of the implementation is simply (standard math).



# Index

- ~GameState
  - GameState, [12](#)
- ~LTexture
  - LTexture, [20](#)
- angle
  - Vector2D, [64](#)
- angleBetween
  - Vector2D, [64](#)
- buttons
  - TitleState.cpp, [112](#)
- calculateDistances
  - Map, [32](#)
- calculateFlowDirections
  - Map, [32](#)
- calculateFlowField
  - Map, [33](#)
- cellCountX
  - Map, [41](#)
- cellCountY
  - Map, [41](#)
- cells
  - Map, [41](#)
- changeState
  - Global.h, [76](#)
  - main.cpp, [101](#)
- close
  - Global.h, [76](#)
  - main.cpp, [102](#)
- cross
  - Vector2D, [65](#)
- deallocateTextures
  - TextureLoader, [55](#)
- destroy
  - LTexture, [21](#)
- dot
  - Vector2D, [65](#)
- draw
  - Map, [34](#)
- drawCell
  - Map, [34](#)
- eButtonSprite
  - LButton, [18](#)
- eButtonSpriteMouseDown
  - LButton, [18](#)
- eButtonSpriteMouseOut
  - LButton, [18](#)
- eButtonSpriteMouseOverMotion
  - LButton, [18](#)
- eButtonSpriteMouseUp
  - LButton, [18](#)
- enter
  - ExitState, [10](#)
  - GameState, [12](#)
  - IntroState, [15](#)
  - MainGameState, [28](#)
  - Part1State, [45](#)
  - Part2State, [50](#)
  - Part3State, [53](#)
  - TitleState, [59](#)
- exit
  - ExitState, [10](#)
  - GameState, [12](#)
  - IntroState, [15](#)
  - MainGameState, [28](#)
  - Part1State, [45](#)
  - Part2State, [50](#)
  - Part3State, [53](#)
  - TitleState, [59](#)
- ExitState, [9](#)
  - enter, [10](#)
  - exit, [10](#)
  - ExitState, [10](#)
  - get, [10](#)
  - handleEvent, [11](#)
  - render, [11](#)
  - sExitState, [12](#)
  - update, [11](#)
- flowDirectionX
  - Map::Cell, [7](#)
- flowDirectionY
  - Map::Cell, [7](#)
- flowDistance
  - Map::Cell, [7](#)
- flowDistanceMax
  - Map, [41](#)
- GameState, [12](#)
  - ~GameState, [12](#)
  - enter, [12](#)
  - exit, [12](#)
  - handleEvent, [13](#)
  - render, [13](#)
  - update, [13](#)
- gButtonSpriteTexture
  - LButton, [19](#)

- gCurrentState
  - main.cpp, 104
- get
  - ExitState, 10
  - IntroState, 16
  - MainGameState, 28
  - Part1State, 45
  - Part2State, 50
  - Part3State, 53
  - TitleState, 59
- getFlowNormal
  - Map, 35
- getHeight
  - LTexture, 21
- getLoadedTextures
  - TextureLoader.cpp, 118
- getNegativeReciprocal
  - Vector2D, 66
- getTargetPos
  - Map, 35
- getTicksNS
  - LTimer, 25
- getWidth
  - LTexture, 21
- gFont
  - Global.h, 78
  - main.cpp, 104
- gFpsTexture
  - main.cpp, 104
- Global, 13
  - kScreenHeight, 13
  - kScreenWidth, 13
- Global.h
  - changeState, 76
  - close, 76
  - gFont, 78
  - gRenderer, 78
  - init, 76
  - loadMedia, 77
  - setNextState, 77
- gNextState
  - main.cpp, 104
- gRenderer
  - Global.h, 78
  - main.cpp, 105
- gWindow
  - main.cpp, 105
- handleEvent
  - ExitState, 11
  - GameState, 13
  - IntroState, 16
  - LButton, 19
  - MainGameState, 29
  - Part1State, 46
  - Part2State, 50
  - Part3State, 53
  - TitleState, 59
- init
  - Global.h, 76
  - main.cpp, 102
- IntroState, 14
  - enter, 15
  - exit, 15
  - get, 16
  - handleEvent, 16
  - IntroState, 15
  - mBackgroundTexture, 17
  - mMessageTexture, 17
  - render, 16
  - sIntroState, 17
  - update, 17
- IntroState.h
  - INTROSTATE\_H, 85
- INTROSTATE\_H
  - IntroState.h, 85
- isCellWall
  - Map, 36
- isClicked
  - LButton, 19
- isInbounds
  - Map, 36
- isPaused
  - LTimer, 25
- isSpawner
  - Map, 37
  - Map::Cell, 8
- isStarted
  - LTimer, 25
- isTarget
  - Map, 38
  - Map::Cell, 8
- isValidPath
  - Map, 39
- isWall
  - Map::Cell, 8
- kButtonCount
  - TitleState.cpp, 112
- kButtonHeight
  - LButton, 19
- kButtonWidth
  - LButton, 19
- keyDownStatus
  - Part1State, 47
- kOriginalSize
  - LTexture, 23
- kScreenFps
  - main.cpp, 105
- kScreenHeight
  - Global, 13
- kScreenWidth
  - Global, 13
- LButton, 17
  - eButtonSprite, 18
  - eButtonSpriteMouseDown, 18

- eButtonSpriteMouseOut, [18](#)
- eButtonSpriteMouseOverMotion, [18](#)
- eButtonSpriteMouseUp, [18](#)
- gButtonSpriteTexture, [19](#)
- handleEvent, [19](#)
- isClicked, [19](#)
- kButtonHeight, [19](#)
- kButtonWidth, [19](#)
- LButton, [18](#)
- mCurrentSprite, [19](#)
- mPosition, [20](#)
- render, [19](#)
- setPosition, [19](#)
- setText, [19](#)
- loadedTextures
  - TextureLoader, [57](#)
- loadFromFile
  - LTexture, [21](#)
- loadFromRenderedText
  - LTexture, [22](#)
- loadMedia
  - Global.h, [77](#)
  - main.cpp, [102](#)
- loadTexture
  - TextureLoader, [55](#)
- LTexture, [20](#)
  - ~LTexture, [20](#)
  - destroy, [21](#)
  - getHeight, [21](#)
  - getWidth, [21](#)
  - kOriginalSize, [23](#)
  - loadFromFile, [21](#)
  - loadFromRenderedText, [22](#)
  - LTexture, [20](#)
  - mHeight, [23](#)
  - mTexture, [24](#)
  - mWidth, [24](#)
  - render, [22](#)
  - setAlpha, [23](#)
  - setBlending, [23](#)
  - setColor, [23](#)
- LTimer, [24](#)
  - getTicksNS, [25](#)
  - isPaused, [25](#)
  - isStarted, [25](#)
  - LTimer, [24](#)
  - mPaused, [26](#)
  - mPausedTicks, [26](#)
  - mStarted, [26](#)
  - mStartTicks, [26](#)
  - pause, [25](#)
  - start, [25](#)
  - stop, [25](#)
  - unpause, [26](#)
- magnitude
  - Vector2D, [66](#)
- main
  - main.cpp, [103](#)
- main.cpp
  - changeState, [101](#)
  - close, [102](#)
  - gCurrentState, [104](#)
  - gFont, [104](#)
  - gFpsTexture, [104](#)
  - gNextState, [104](#)
  - gRenderer, [105](#)
  - gWindow, [105](#)
  - init, [102](#)
  - kScreenFps, [105](#)
  - loadMedia, [102](#)
  - main, [103](#)
  - setNextState, [103](#)
- MainGameState, [27](#)
  - enter, [28](#)
  - exit, [28](#)
  - get, [28](#)
  - handleEvent, [29](#)
  - MainGameState, [28](#)
  - mBackgroundTexture, [30](#)
  - mMessageTexture, [30](#)
  - render, [29](#)
  - sMainGameState, [30](#)
  - update, [29](#)
- Map, [30](#)
  - calculateDistances, [32](#)
  - calculateFlowDirections, [32](#)
  - calculateFlowField, [33](#)
  - cellCountX, [41](#)
  - cellCountY, [41](#)
  - cells, [41](#)
  - draw, [34](#)
  - drawCell, [34](#)
  - flowDistanceMax, [41](#)
  - getFlowNormal, [35](#)
  - getTargetPos, [35](#)
  - isCellWall, [36](#)
  - isInbounds, [36](#)
  - isSpawner, [37](#)
  - isTarget, [38](#)
  - isValidPath, [39](#)
  - Map, [31](#)
  - PIXELS\_PER\_CELL, [42](#)
  - setCellWall, [39](#)
  - setSpawner, [40](#)
  - setTarget, [40](#)
  - textureCellArrowDown, [42](#)
  - textureCellArrowLeft, [42](#)
  - textureCellArrowRight, [42](#)
  - textureCellArrowUp, [42](#)
  - textureCellEmpty, [42](#)
  - textureCellSpawner, [42](#)
  - textureCellTarget, [42](#)
  - textureCellWall, [43](#)
- map
  - Part1State, [47](#)
- Map::Cell, [7](#)

- flowDirectionX, [7](#)
  - flowDirectionY, [7](#)
  - flowDistance, [7](#)
  - isSpawner, [8](#)
  - isTarget, [8](#)
  - isWall, [8](#)
  - x, [8](#)
  - y, [8](#)
- mBackgroundTexture
  - IntroState, [17](#)
  - MainGameState, [30](#)
  - Part1State, [47](#)
  - Part2State, [51](#)
  - Part3State, [54](#)
  - TitleState, [60](#)
- mCurrentSprite
  - LButton, [19](#)
- mHeight
  - LTexture, [23](#)
- mMessageTexture
  - IntroState, [17](#)
  - MainGameState, [30](#)
  - Part1State, [47](#)
  - Part2State, [51](#)
  - Part3State, [54](#)
  - TitleState, [60](#)
- mouseDownStatus
  - Part1State, [48](#)
- mPaused
  - LTimer, [26](#)
- mPausedTicks
  - LTimer, [26](#)
- mPosition
  - LButton, [20](#)
- mStarted
  - LTimer, [26](#)
- mStartTicks
  - LTimer, [26](#)
- mTexture
  - LTexture, [24](#)
- mWidth
  - LTexture, [24](#)
- normalize
  - Vector2D, [66](#)
- operator+
  - Vector2D, [68](#), [69](#)
- operator+=
  - Vector2D, [69](#)
- operator-
  - Vector2D, [70](#)
- operator-=
  - Vector2D, [70](#), [71](#)
- operator/
  - Vector2D, [71](#)
- operator/=
  - Vector2D, [72](#)
- operator\*
  - Vector2D, [67](#)
- operator\*=
  - Vector2D, [68](#)
- Part1State, [43](#)
  - enter, [45](#)
  - exit, [45](#)
  - get, [45](#)
  - handleEvent, [46](#)
  - keyDownStatus, [47](#)
  - map, [47](#)
  - mBackgroundTexture, [47](#)
  - mMessageTexture, [47](#)
  - mouseDownStatus, [48](#)
  - Part1State, [45](#)
  - render, [46](#)
  - sPart1State, [48](#)
  - update, [47](#)
- Part2State, [48](#)
  - enter, [50](#)
  - exit, [50](#)
  - get, [50](#)
  - handleEvent, [50](#)
  - mBackgroundTexture, [51](#)
  - mMessageTexture, [51](#)
  - Part2State, [49](#)
  - render, [50](#)
  - sPart2State, [51](#)
  - update, [51](#)
- Part3State, [51](#)
  - enter, [53](#)
  - exit, [53](#)
  - get, [53](#)
  - handleEvent, [53](#)
  - mBackgroundTexture, [54](#)
  - mMessageTexture, [54](#)
  - Part3State, [53](#)
  - render, [54](#)
  - sPart3State, [54](#)
  - update, [54](#)
- pause
  - LTimer, [25](#)
- PIXELS\_PER\_CELL
  - Map, [42](#)
- render
  - ExitState, [11](#)
  - GameState, [13](#)
  - IntroState, [16](#)
  - LButton, [19](#)
  - LTexture, [22](#)
  - MainGameState, [29](#)
  - Part1State, [46](#)
  - Part2State, [50](#)
  - Part3State, [54](#)
  - TitleState, [60](#)
- setAlpha
  - LTexture, [23](#)



- setBlending
  - LTexture, 23
- setCellWall
  - Map, 39
- setColor
  - LTexture, 23
- setNextState
  - Global.h, 77
  - main.cpp, 103
- setPosition
  - LButton, 19
- setSpawner
  - Map, 40
- setTarget
  - Map, 40
- setText
  - LButton, 19
- sExitState
  - ExitState, 12
- sIntroState
  - IntroState, 17
- sMainGameState
  - MainGameState, 30
- sPart1State
  - Part1State, 48
- sPart2State
  - Part2State, 51
- sPart3State
  - Part3State, 54
- src/include/Global.h, 75, 78
- src/include/LTimer.h, 79
- src/include/map/Map.h, 79, 81
- src/include/states/ExitState.h, 82, 83
- src/include/states/GameState.h, 83, 84
- src/include/states/IntroState.h, 84, 85
- src/include/states/parts/MainGameState.h, 86, 87
- src/include/states/parts/Part1State.h, 88, 89
- src/include/states/parts/Part2State.h, 89, 90
- src/include/states/parts/Part3State.h, 91, 92
- src/include/states/TitleState.h, 93, 94
- src/include/ui/LButton.h, 94, 95
- src/include/ui/LTexture.h, 96
- src/include/util/TextureLoader.h, 97, 98
- src/include/util/Vector2D.h, 99, 100
- src/LTimer.cpp, 100
- src/main.cpp, 101
- src/map/Map.cpp, 105
- src/states/ExitState.cpp, 106
- src/states/IntroState.cpp, 107, 108
- src/states/parts/MainGameState.cpp, 109
- src/states/parts/Part1State.cpp, 109
- src/states/parts/Part2State.cpp, 110
- src/states/parts/Part3State.cpp, 111
- src/states/TitleState.cpp, 112, 113
- src/ui/LButton.cpp, 115
- src/ui/LTexture.cpp, 115, 116
- src/util/TextureLoader.cpp, 118
- src/util/Vector2D.cpp, 119
- start
  - LTimer, 25
- sTitleState
  - TitleState, 60
- stop
  - LTimer, 25
- TEXTURE\_PATH
  - TextureLoader, 57
- textureCellArrowDown
  - Map, 42
- textureCellArrowLeft
  - Map, 42
- textureCellArrowRight
  - Map, 42
- textureCellArrowUp
  - Map, 42
- textureCellEmpty
  - Map, 42
- textureCellSpawner
  - Map, 42
- textureCellTarget
  - Map, 42
- textureCellWall
  - Map, 43
- TextureLoader, 54
  - deallocateTextures, 55
  - loadedTextures, 57
  - loadTexture, 55
  - TEXTURE\_PATH, 57
- TextureLoader.cpp
  - getLoadedTextures, 118
- TitleState, 57
  - enter, 59
  - exit, 59
  - get, 59
  - handleEvent, 59
  - mBackgroundTexture, 60
  - mMessageTexture, 60
  - render, 60
  - sTitleState, 60
  - TitleState, 58
  - update, 60
- TitleState.cpp
  - buttons, 112
  - kButtonCount, 112
- unpause
  - LTimer, 26
- update
  - ExitState, 11
  - GameState, 13
  - IntroState, 17
  - MainGameState, 29
  - Part1State, 47
  - Part2State, 51
  - Part3State, 54
  - TitleState, 60

Vector2D, [61](#)  
  angle, [64](#)  
  angleBetween, [64](#)  
  cross, [65](#)  
  dot, [65](#)  
  getNegativeReciprocal, [66](#)  
  magnitude, [66](#)  
  normalize, [66](#)  
  operator+, [68](#), [69](#)  
  operator+=, [69](#)  
  operator-, [70](#)  
  operator-=, [70](#), [71](#)  
  operator/, [71](#)  
  operator/=: [72](#)  
  operator\*, [67](#)  
  operator\*=: [68](#)  
  Vector2D, [62](#), [63](#)  
  x, [73](#)  
  y, [73](#)

x  
  Map::Cell, [8](#)  
  Vector2D, [73](#)

y  
  Map::Cell, [8](#)  
  Vector2D, [73](#)