



=1



# Visualizer

1.0.0

Generated by Doxygen 1.10.0



# Chapter 1

## Crossy Road clone

CS202 group project.

### 1.1 Dependencies

#### 1.1.1 Required

- GCC 11
- Cmake
- Makefile

#### 1.1.2 Automated downloads

- Raylib
- GLFW
- cppyaml



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

PASSETS	.....	??
PATB	.....	??
REQUEST	.....	??
VECTOR2D	.....	??
YAML_FILE	.....	??
Opens and interacts with YAML files	.....	??





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

Action . . . . .	??
CloseAction . . . . .	??
PacketAction . . . . .	??
Request . . . . .	??
changeInfRequest . . . . .	??
loseRequest . . . . .	??
changeImageAction . . . . .	??
moveChunksAction . . . . .	??
moveEntityAction . . . . .	??
moveObjectAction . . . . .	??
resizeAction . . . . .	??
ARGS . . . . .	??
CountDown . . . . .	??
Frame . . . . .	??
Container . . . . .	??
ButtonImage . . . . .	??
Interface . . . . .	??
Chunk . . . . .	??
Game . . . . .	??
Object . . . . .	??
Visual . . . . .	??
KeyStroke . . . . .	??
RandomEngine . . . . .	??
Window . . . . .	??



## Chapter 4

# Class Index

### 4.1 Class List

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

Action	Manages the way an action is executed . . . . .	??
ARGS	Stores request information . . . . .	??
ButtonImage	Manages the appearance and behavior of a button . . . . .	??
changeImageAction	Changes display image of container . . . . .	??
changeInfRequest	. . . . .	??
Chunk	Manages the spawning of chunks and how entities interact with them . . . . .	??
CloseAction	Manages the closing of the application . . . . .	??
Container	Holds specific entities and their behavior . . . . .	??
CountDown	Count the time a playthrough takes . . . . .	??
Frame	Position and size of object on screen . . . . .	??
Game	. . . . .	??
Interface	Where user can interact with the game . . . . .	??
KeyStroke	Manages the link between a key and the actions it performs . . . . .	??
loseRequest	Request sent when the player loses . . . . .	??
moveChunksAction	. . . . .	??
moveEntityAction	. . . . .	??
moveObjectAction	Manages the features of a movement, including which object, speed, direction etc . . . . .	??
Object	. . . . .	??
PacketAction	Organize selected actions into a package . . . . .	??
RandomEngine	. . . . .	??
Request	Sends information to a higher, relevant entity . . . . .	??

<a href="#">resizeAction</a>	
Manages the resizing of the window . . . . .	??
<a href="#">Visual</a>	
Images displayed on screen . . . . .	??
<a href="#">Window</a>	. . . . . ??

# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	??
src/action/include/action.hpp	??
src/action/include/request.hpp	??
src/action/src/action.cpp	??
src/action/src/args.cpp	??
src/action/src/request.cpp	??
src/action/src/request/changeinf.cpp	??
src/action/src/request/lose.cpp	??
src/button/include/button.hpp	??
src/button/src/arithmetic.cpp	??
src/button/src/constructor.cpp	??
src/button/src/destructor.cpp	??
src/chunk/include/chunk.hpp	??
src/chunk/src/action.cpp	??
src/chunk/src/arithmetic.cpp	??
src/chunk/src/constructor.cpp	??
src/chunk/src/destructor.cpp	??
src/chunk/src/action/moveentity.cpp	??
src/container/include/container.hpp	??
src/container/src/arithmetic.cpp	??
src/container/src/constructor.cpp	??
src/container/src/destructor.cpp	??
src/container/src/overlap.cpp	??
src/container/src/action/changesprite.cpp	??
src/frame/include/frame.hpp	??
src/frame/src/arithmetic.cpp	??
src/frame/src/constructor.cpp	??
src/frame/src/destructor.cpp	??
src/frame/src/family.cpp	??
src/game/include/game.hpp	??
src/game/src/action.cpp	??
src/game/src/arithmetic.cpp	??
src/game/src/constructor.cpp	??
src/game/src/destructor.cpp	??
src/game/src/action/movechunk.cpp	??

src/interface/include/interface.hpp	??
src/interface/src/action.cpp	??
src/interface/src/arithmic.cpp	??
src/interface/src/constructor.cpp	??
src/interface/src/destructor.cpp	??
src/interface/src/action/moveobject.cpp	??
src/object/include/object.hpp	??
src/object/src/arithmic.cpp	??
src/object/src/constructor.cpp	??
src/object/src/destructor.cpp	??
src/utls/include/countdown.hpp	??
src/utls/include/file.hpp	??
src/utls/include/keystroke.hpp	??
src/utls/include/random.hpp	??
src/utls/include/vector.hpp	??
src/utls/include/const/datatype.hpp	??
src/utls/include/const/request.hpp	??
src/utls/include/const/path/assets.hpp	??
src/utls/include/const/path/atb.hpp	??
src/utls/src/constant.cpp	??
src/utls/src/countdown.cpp	??
src/utls/src/file.cpp	??
src/utls/src/keystroke.cpp	??
src/utls/src/random.cpp	??
src/utls/src/request.cpp	??
src/utls/src/vector.cpp	??
src/visual/include/visual.hpp	??
src/visual/src/arithmic.cpp	??
src/visual/src/constructor.cpp	??
src/visual/src/destructor.cpp	??
src/window/include/window.hpp	??
src/window/src/acting.cpp	??
src/window/src/constructor.cpp	??
src/window/src/destructor.cpp	??
src/window/src/interface.cpp	??
src/window/src/running.cpp	??
src/window/src/UI.cpp	??
src/window/src/wincontent.cpp	??
src/window/src/action/close.cpp	??
src/window/src/action/resize.cpp	??

## Chapter 6

# Namespace Documentation

### 6.1 PASSETS Namespace Reference

#### Variables

- `const std::string GRAPHIC_ = "assets/graphics/"`
- `const std::string SOUND_ = "assets/sounds/"`
- `const std::string FONT_ = "assets/fonts/"`

#### 6.1.1 Variable Documentation

##### 6.1.1.1 FONT\_

```
const std::string PASSETS::FONT_ = "assets/fonts/" [extern]
```

Definition at line 6 of file [constant.cpp](#).

##### 6.1.1.2 GRAPHIC\_

```
const std::string PASSETS::GRAPHIC_ = "assets/graphics/" [extern]
```

Definition at line 4 of file [constant.cpp](#).

##### 6.1.1.3 SOUND\_

```
const std::string PASSETS::SOUND_ = "assets/sounds/" [extern]
```

Definition at line 5 of file [constant.cpp](#).



## 6.2 PATB Namespace Reference

### Variables

- `const std::string ATB_ = "atb/"`
- `const std::string WINDOW_ = "atb/window/"`
- `const std::string INTERFACE_ = "atb/interface/"`
- `const std::string BUTTON_ = "atb/button/"`
- `const std::string CONTAINER_ = "atb/container/"`
- `const std::string OBJECT_ = "atb/object/"`
- `const std::string MAP_ = "atb/map/"`
- `const std::string CHUNK_ = "atb/chunk/"`
- `const std::string BLOCK_ = "atb/block/"`
- `const std::string ENTITY_ = "atb/entity/"`

### 6.2.1 Variable Documentation

#### 6.2.1.1 ATB\_

```
const std::string PATB::ATB_ = "atb/" [extern]
```

Definition at line 8 of file [constant.cpp](#).

#### 6.2.1.2 BLOCK\_

```
const std::string PATB::BLOCK_ = "atb/block/" [extern]
```

Definition at line 16 of file [constant.cpp](#).

#### 6.2.1.3 BUTTON\_

```
const std::string PATB::BUTTON_ = "atb/button/" [extern]
```

Definition at line 11 of file [constant.cpp](#).

#### 6.2.1.4 CHUNK\_

```
const std::string PATB::CHUNK_ = "atb/chunk/" [extern]
```

Definition at line 15 of file [constant.cpp](#).

#### 6.2.1.5 CONTAINER\_

```
const std::string PATB::CONTAINER_ = "atb/container/" [extern]
```

Definition at line 12 of file [constant.cpp](#).

#### 6.2.1.6 ENTITY\_

```
const std::string PATB::ENTITY_ = "atb/entity/" [extern]
```

Definition at line 17 of file [constant.cpp](#).

#### 6.2.1.7 INTERFACE\_

```
const std::string PATB::INTERFACE_ = "atb/interface/" [extern]
```

Definition at line 10 of file [constant.cpp](#).

#### 6.2.1.8 MAP\_

```
const std::string PATB::MAP_ = "atb/map/" [extern]
```

Definition at line 14 of file [constant.cpp](#).

#### 6.2.1.9 OBJECT\_

```
const std::string PATB::OBJECT_ = "atb/object/" [extern]
```

Definition at line 13 of file [constant.cpp](#).

#### 6.2.1.10 WINDOW\_

```
const std::string PATB::WINDOW_ = "atb/window/" [extern]
```

Definition at line 9 of file [constant.cpp](#).

## 6.3 REQUEST Namespace Reference

### Enumerations

- enum [ID](#) {  
    [INVALID](#), [NONE](#), [CHANGE\\_INF](#), [DELAY](#),  
    [LOSE](#) }

### 6.3.1 Enumeration Type Documentation

#### 6.3.1.1 ID

```
enum REQUEST::ID
```

## Enumerator

INVALID	
NONE	
CHANGE_INF	
DELAY	
LOSE	

Definition at line 6 of file [request.hpp](#).

```

00007 {
00008     INVALID,
00009     NONE,
00010     CHANGE_INF,
00011     DELAY,
00012     LOSE,
00013 };

```

## 6.4 VECTOR2D Namespace Reference

### Functions

- float [getAngle](#) (fPoint v1)
- float [getAngle](#) (fPoint v1, fPoint v2)

### 6.4.1 Function Documentation

#### 6.4.1.1 getAngle() [1/2]

```

float VECTOR2D::getAngle (
    fPoint v1 )

```

Definition at line 5 of file [vector.cpp](#).

```

00006 {
00007     // arctan(y / x)
00008     return atan2(v1[1], v1[0]);
00009 }

```

#### 6.4.1.2 getAngle() [2/2]

```

float VECTOR2D::getAngle (
    fPoint v1,
    fPoint v2 )

```

Definition at line 16 of file [vector.cpp](#).

```

00017 {
00018     // angle between 2 vector
00019     // v1 * v2 = |v1| * |v2| * cos(angle)
00020
00021     float dot = v1[0] * v2[0] + v1[1] * v2[1];
00022     float abs1 = sqrt(sqr(v1[0]) + sqr(v1[1]));
00023     float abs2 = sqrt(sqr(v2[0]) + sqr(v2[1]));
00024     return acos(dot / (abs1 * abs2));
00025 }

```

## 6.5 YAML\_FILE Namespace Reference

opens and interacts with YAML files

### Functions

- bool [isFile](#) (std::string path)
- YAML::Node [readFile](#) (std::string path)
- bool [writeFile](#) (std::string path, YAML::Node content)

### 6.5.1 Detailed Description

opens and interacts with YAML files

### 6.5.2 Function Documentation

#### 6.5.2.1 isFile()

```
bool YAML_FILE::isFile (  
    std::string path )
```

Definition at line 5 of file [file.cpp](#).

```
00006 {  
00007     // return true if file exists  
00008  
00009     std::ifstream fin(path);  
00010     return fin.good();  
00011 }
```

#### 6.5.2.2 readFile()

```
YAML::Node YAML_FILE::readFile (  
    std::string path )
```

Definition at line 13 of file [file.cpp](#).

```
00014 {  
00015     // return YAML::Node from file  
00016  
00017     YAML::Node node;  
00018     try  
00019     {  
00020         node = YAML::LoadFile(path);  
00021     }  
00022     catch (YAML::BadFile& e)  
00023     {  
00024         std::cout << "Error: " << e.what() << std::endl;  
00025     }  
00026     return node;  
00027 }
```

#### 6.5.2.3 writeFile()

```
bool YAML_FILE::writeFile (  
    std::string path,  
    YAML::Node content )
```



## Chapter 7

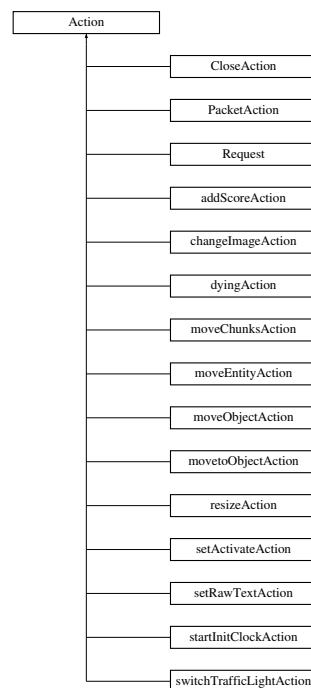
# Class Documentation

### 7.1 Action Class Reference

manages the way an action is executed

```
#include <action.hpp>
```

Inheritance diagram for Action:



#### Public Member Functions

- [Action](#) ()
- [Action](#) ([Action](#) \*)
- virtual [~Action](#) ()=default
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual void [execute](#) ()
- virtual [Action](#) \* [clone](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.1.1 Detailed Description

manages the way an action is executed

Definition at line 31 of file [action.hpp](#).

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 Action() [1/2]

```
Action::Action ( )
```

Definition at line 6 of file [action.cpp](#).

```
00007 {  
00008 }
```

#### 7.1.2.2 Action() [2/2]

```
Action::Action (  
    Action * action )
```

Definition at line 10 of file [action.cpp](#).

```
00011 {  
00012 }
```

#### 7.1.2.3 ~Action()

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

### 7.1.3 Member Function Documentation

#### 7.1.3.1 clone()

```
Action * Action::clone ( ) [virtual]
```

Reimplemented in [PacketAction](#), [Request](#), [changeInfRequest](#), [loseRequest](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 29 of file [action.cpp](#).

```
00030 {  
00031     return this;  
00032 }
```

#### 7.1.3.2 execute()

```
void Action::execute ( ) [virtual]
```

Reimplemented in [CloseAction](#), [resizeAction](#), [PacketAction](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 25 of file [action.cpp](#).

```
00026 {  
00027 }
```

### 7.1.3.3 getArgs()

`ARGS & Action::getArgs ( ) [virtual]`

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.1.3.4 isPackage()

`bool Action::isPackage ( ) [virtual]`

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.1.3.5 isRequest()

`int Action::isRequest ( ) [virtual]`

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {  
00017     return 0;  
00018 }
```

### 7.1.3.6 unpack()

`std::vector< Action * > Action::unpack ( ) [virtual]`

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

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

- [src/action/include/action.hpp](#)
- [src/action/src/action.cpp](#)

## 7.2 ARGS Struct Reference

stores request information

```
#include <action.hpp>
```



## Public Member Functions

- [ARGS](#) ()=default
- [~ARGS](#) ()=default
- std::string [getInterfaceName](#) ()

## Public Attributes

- std::vector< std::string > [str](#)
- std::vector< int > [num](#)
- std::vector< void \* > [addr](#)

## 7.2.1 Detailed Description

stores request information

Definition at line 13 of file [action.hpp](#).

## 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 ARGS()

```
ARGS::ARGS ( ) [default]
```

### 7.2.2.2 ~ARGS()

```
ARGS::~~ARGS ( ) [default]
```

## 7.2.3 Member Function Documentation

### 7.2.3.1 getInterfaceName()

```
std::string ARGS::getInterfaceName ( )
```

Definition at line 3 of file [args.cpp](#).

```
00004 {  
00005     return str[0];  
00006 }
```

## 7.2.4 Member Data Documentation

### 7.2.4.1 addr

```
std::vector<void*> ARGS::addr
```

Definition at line 17 of file [action.hpp](#).

#### 7.2.4.2 num

```
std::vector<int> ARGS::num
```

Definition at line 16 of file [action.hpp](#).

#### 7.2.4.3 str

```
std::vector<std::string> ARGS::str
```

Definition at line 15 of file [action.hpp](#).

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

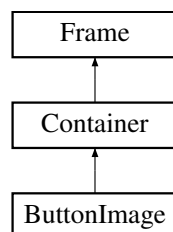
- [src/action/include/action.hpp](#)
- [src/action/src/args.cpp](#)

## 7.3 ButtonImage Class Reference

manages the appearance and behavior of a button

```
#include <button.hpp>
```

Inheritance diagram for ButtonImage:



### Public Member Functions

- [ButtonImage](#) ([Frame](#) \*parent, Rectangle relative)
- [~ButtonImage](#) ()
- void [draw](#) ()
- [PacketAction](#) \* [react](#) ()
- void [changeIndex](#) (int newIndex)
- void [changePosition](#) (Rectangle change)
- bool [isClicked](#) () const
- bool [isPressing](#) () const
- int [getClicked](#) ()
- std::string [linkContent](#) (std::string)
- std::string [linkContentAbsolute](#) (std::string)
- std::string [getName](#) ()
- void [setProbability](#) (int)
- int [getProbability](#) ()
- void [chooseSprite](#) (int)

- choose a specific sprite from a vector of sprites*
- void [chooseImage](#) (int)
  - choose the state of the sprite*
- void [chooseImage](#) (int, int)
  - choose the state of the sprite*
- void [nextImage](#) ()
  - move to next state of the sprite*
- void [prevImage](#) ()
  - move to previous state of the sprite*
- void [nextSprite](#) ()
  - move to the next sprite*
- void [prevSprite](#) ()
  - move to the previous sprite*
- bool [isOverlapping](#) (fPoint)
- bool [isOverlapping](#) (Rectangle)
- bool [isOverlapping](#) (Container \*)
- float [OverlappingArea](#) (Rectangle)
- float [OverlappingArea](#) (Container \*)
- void [show](#) ()
- void [hide](#) ()
- void [toggleVisibility](#) ()
- bool [isVisible](#) ()
- int [getInstanceId](#) ()
- virtual Action \* [getRuntimeEvent](#) ()
- void [plug](#) (Frame \*par, fRect rel)
  - attach a frame to a parent by relative position*
- void [plug](#) (Frame \*par)
  - attach a frame to a parent by old relative position*
- void [unplug](#) ()
  - detach a frame from its parent*
- void [moveTo](#) (fPoint rel)
- void [moveTo](#) (int x, int y)
- void [moveCenterTo](#) (fPoint rel)
- void [moveCenterTo](#) (int x, int y)
- void [moveBy](#) (fPoint rel)
- void [moveBy](#) (int, int)
- void [resize](#) (fPoint rel)
- void [resize](#) (int w, int h)
- const Rectangle & [getFrame](#) () const
- const fRect & [getRelative](#) () const
- Frame \* [getParent](#) ()
- void [setRelative](#) (fRect rel)
- const fPoint & [getCenter](#) () const
- const float & [getX](#) () const
- const float & [getY](#) () const
- const float & [getW](#) () const
- const float & [getH](#) () const
- operator Rectangle () const
- operator fRect () const
- operator iRect () const

## Protected Member Functions

- void [loadEvent](#) (YAML::Node node)
- bool [loadName](#) (YAML::Node node)
- void [loadSprites](#) (YAML::Node node)
- void [loadFocus](#) (YAML::Node node)
- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) (Frame \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) (Frame \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

### 7.3.1 Detailed Description

manages the appearance and behavior of a button

Definition at line 18 of file [button.hpp](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 ButtonImage()

```
ButtonImage::ButtonImage (
    Frame * parent,
    Rectangle relative )
```

Definition at line 5 of file [constructor.cpp](#).

```
00005                                     : Container(parent, rel)
00006 {
00007     // set default
00008     this->chooseImage(0, this->tmpPath);
00009     this->color = WHITE;
00010     this->pressing = false;
00011     this->isHover = false;
00012     this->clicked = false;
00013
00014     this->releaseID = -1;
00015     this->hoverID = -1;
00016     this->pressingID = -1;
00017     this->clickedID = -1;
00018 }
```

#### 7.3.2.2 ~ButtonImage()

```
ButtonImage::~~ButtonImage ( )
```

Definition at line 3 of file [destructor.cpp](#).

```
00004 {
00005     for(auto &action : actions)
00006     {
00007         delete action;
00008     }
00009 }
```

### 7.3.3 Member Function Documentation

#### 7.3.3.1 addSubframe()

```
void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]
```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

##### Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```
00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }
```

#### 7.3.3.2 beginUpdate()

```
void Frame::beginUpdate ( ) [protected], [inherited]
```

Definition at line 113 of file [family.cpp](#).

```
00114 {
00115     mtx.lock();
00116 }
```

#### 7.3.3.3 changeIndex()

```
void ButtonImage::changeIndex (
    int newIndex )
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     tmpPath = newIndex;
00057 }
```

#### 7.3.3.4 changePosition()

```
void ButtonImage::changePosition (
    Rectangle change )
```

#### 7.3.3.5 chooseImage() [1/2]

```
void Container::chooseImage (
    int index ) [inherited]
```

choose the state of the sprite

Definition at line 231 of file [constructor.cpp](#).

```
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
```

### 7.3.3.6 chooseImage() [2/2]

```
void Container::chooseImage (
    int index,
    int index2 ) [inherited]
```

choose the state of the sprite

Definition at line 238 of file [constructor.cpp](#).

```
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
```

### 7.3.3.7 chooseSprite()

```
void Container::chooseSprite (
    int index ) [inherited]
```

choose a specific sprite from a vector of sprites

Definition at line 224 of file [constructor.cpp](#).

```
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
```

### 7.3.3.8 draw()

```
void ButtonImage::draw ( ) [virtual]
```

Reimplemented from [Container](#).

Definition at line 7 of file [arithmetic.cpp](#).

```
00007 {
00008     if(!isVisible()) return;
00009     this->Container::draw();
00010 }
```

### 7.3.3.9 endUpdate()

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {
00120     mtx.unlock();
00121 }
```

### 7.3.3.10 getCenter()

const [fPoint](#) & Frame::getCenter ( ) const [inherited]

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139
00140     return resu;
00141 }
```

### 7.3.3.11 getClicked()

int ButtonImage::getClicked ( )

Definition at line 59 of file [arithmetic.cpp](#).

```
00060 {
00061     return tmpPath;
00062 }
```

### 7.3.3.12 getFrame()

const [Rectangle](#) & Frame::getFrame ( ) const [inherited]

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {
00107     std::lock_guard<std::mutex> lock(mtx);
00108     return frame;
00109 }
```

### 7.3.3.13 getH()

const float & Frame::getH ( ) const [inherited]

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {
00163     std::lock_guard<std::mutex> lock(mtx);
00164     return frame.height;
00165 }
```

### 7.3.3.14 getInstanceId()

int Container::getInstanceId ( ) [inherited]

Definition at line 31 of file [arithmetic.cpp](#).

```
00032 {
00033     return instance_id;
00034 }
```

### 7.3.3.15 getName()

std::string Container::getName ( ) [inherited]

Definition at line 275 of file [constructor.cpp](#).

```
00276 {  
00277     return name;  
00278 }
```

### 7.3.3.16 getParent()

Frame \* Frame::getParent ( ) [inherited]

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```

### 7.3.3.17 getProbability()

int Container::getProbability ( ) [inherited]

Definition at line 285 of file [constructor.cpp](#).

```
00286 {  
00287     return probability;  
00288 }
```

### 7.3.3.18 getRelative()

const fRect & Frame::getRelative ( ) const [inherited]

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {  
00113     std::lock_guard<std::mutex> lock(mtx);  
00114     return relative;  
00115 }
```

### 7.3.3.19 getRuntimeEvent()

Action \* Container::getRuntimeEvent ( ) [virtual], [inherited]

Reimplemented in [Chunk](#), [Game](#), and [Interface](#).

Definition at line 41 of file [arithmetic.cpp](#).

```
00042 {  
00043     return nullptr;  
00044 }
```

### 7.3.3.20 getW()

const float & Frame::getW ( ) const [inherited]

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {  
00157     std::lock_guard<std::mutex> lock(mtx);  
00158     return frame.width;  
00159 }
```



**7.3.3.21 getX()**

```
const float & Frame::getX ( ) const [inherited]
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {
00145     std::lock_guard<std::mutex> lock(mtx);
00146     return frame.x;
00147 }
```

**7.3.3.22 getY()**

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {
00151     std::lock_guard<std::mutex> lock(mtx);
00152     return frame.y;
00153 }
```

**7.3.3.23 hide()**

```
void Container::hide ( ) [inherited]
```

Definition at line 16 of file [arithmetic.cpp](#).

```
00017 {
00018     visible = false;
00019 }
```

**7.3.3.24 isClicked()**

```
bool ButtonImage::isClicked ( ) const
```

Definition at line 64 of file [arithmetic.cpp](#).

```
00064 {
00065     return this->clicked;
00066 }
```

**7.3.3.25 isOverlapping() [1/3]**

```
bool Container::isOverlapping (
    Container * container ) [inherited]
```

Definition at line 16 of file [overlap.cpp](#).

```
00017 {
00018     Rectangle rec = getFrame();
00019     Rectangle rec2 = container->getFrame();
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
    rec2.height && rec.y + rec.height >= rec2.y);
00021 }
```

**7.3.3.26 isOverlapping() [2/3]**

```
bool Container::isOverlapping (
    fPoint point ) [inherited]
```

Definition at line 3 of file [overlap.cpp](#).

```
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
    rec.y + rec.height);
00007
00008 }
```

**7.3.3.27 isOverlapping() [3/3]**

```
bool Container::isOverlapping (
    Rectangle rec ) [inherited]
```

Definition at line 10 of file [overlap.cpp](#).

```
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
    rec2.height && rec.y + rec.height >= rec2.y);
00014 }
```

**7.3.3.28 isPressing()**

```
bool ButtonImage::isPressing ( ) const
```

**7.3.3.29 isroot()**

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

**7.3.3.30 isVisible()**

```
bool Container::isVisible ( ) [inherited]
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     return visible;
00029 }
```

**7.3.3.31 linkContent()**

```
std::string ButtonImage::linkContent (
    std::string path ) [virtual]
```

Reimplemented from [Container](#).

Definition at line 20 of file [constructor.cpp](#).

```
00021 {
00022     return linkContentAbsolute(PATB::BUTTON_ + path);
00023 }
```

### 7.3.3.32 linkContentAbsolute()

```
std::string ButtonImage::linkContentAbsolute (
    std::string path ) [virtual]
```

Reimplemented from [Container](#).

Definition at line 25 of file [constructor.cpp](#).

```
00026 {
00027     YAML::Node node = YAML_FILE::readFile(path);
00028     if(!loadName(node)) return "";
00029
00030     if(node["textures"])
00031     {
00032         loadSprites(node["textures"]);
00033         chooseImage(0, 0);
00034     }
00035     if(node["events"])
00036     {
00037         loadEvent(node["events"]);
00038     }
00039
00040     return getName();
00041 }
```

### 7.3.3.33 loadEvent()

```
void ButtonImage::loadEvent (
    YAML::Node node ) [protected]
```

Definition at line 43 of file [constructor.cpp](#).

```
00044 {
00045     if(node["hover"])
00046     {
00047         for(auto sprite : node["hover"]["sprite"])
00048         {
00049             iPoint p;
00050             int delay = 0;
00051             p[0] = sprite[0].as<int>();
00052             p[1] = sprite[1].as<int>();
00053             if(p.size() >= 3)
00054                 delay = sprite[2].as<int>();
00055             actions.push_back(new changeImageAction(this, p));
00056         }
00057         this->hoverID = actions.size() - 1;
00058     }
00059
00060     if(node["release"])
00061     {
00062         for(auto sprite : node["release"]["sprite"])
00063         {
00064             iPoint p;
00065             int delay = 0;
00066             p[0] = sprite[0].as<int>();
00067             p[1] = sprite[1].as<int>();
00068             if(p.size() >= 3)
00069                 delay = sprite[2].as<int>();
00070             actions.push_back(new changeImageAction(this, p));
00071         }
00072         this->releaseID = actions.size() - 1;
00073     }
00074
00075     if(node["clicked"])
00076     {
00077         for(auto sprite : node["clicked"]["sprite"])
00078         {
00079             iPoint p;
00080             int delay = 0;
00081             p[0] = sprite[0].as<int>();
00082             p[1] = sprite[1].as<int>();
00083             if(p.size() >= 3)
00084                 delay = sprite[2].as<int>();
00085             actions.push_back(new changeImageAction(this, p));
00086         }
00087         this->clickedID = actions.size() - 1;
00088     }
00089 }
```

```

00090     }
00091
00092     if(node["pressing"])
00093     {
00094         for(auto sprite : node["pressing"]["sprite"])
00095         {
00096             iPoint p;
00097             int delay = 0;
00098             p[0] = sprite[0].as<int>();
00099             p[1] = sprite[1].as<int>();
00100             if(p.size() >= 3)
00101                 delay = sprite[2].as<int>();
00102             actions.push_back(new changeImageAction(this, p));
00103         }
00104         this->pressingID = actions.size() - 1;
00105     }
00106 }

```

### 7.3.3.34 loadFocus()

```

void Container::loadFocus (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 218 of file [constructor.cpp](#).

```

00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }

```

### 7.3.3.35 loadName()

```

bool Container::loadName (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 111 of file [constructor.cpp](#).

```

00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }

```

### 7.3.3.36 loadSprites()

```

void Container::loadSprites (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 122 of file [constructor.cpp](#).

```

00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])

```

```

00141     {
00142         float x, y, w, h;
00143         int repeat = 1;
00144         int gapX = 0;
00145         int gapY = 0;
00146
00147         int dx = 1;
00148         int dy = 1;
00149
00150         if(img["x"])
00151             x = img["x"].as<float>() / 100.0;
00152         else x = 0;
00153         if(img["y"])
00154             y = img["y"].as<float>() / 100.0;
00155         else y = 0;
00156         if(img["w"])
00157             w = img["w"].as<float>() / 100.0;
00158         else w = 1;
00159         if(img["h"])
00160             h = img["h"].as<float>() / 100.0;
00161         else h = 1;
00162         if(img["repeat"])
00163             repeat = img["repeat"].as<int>();
00164         if(img["gapX"])
00165             gapX = img["gapX"].as<int>();
00166         if(img["gapY"])
00167             gapY = img["gapY"].as<int>();
00168
00169         if(img["dx"])
00170             dx = img["dx"].as<int>();
00171         if(dx < 0) dx = -1;
00172         else dx = 1;
00173
00174         if(img["dy"])
00175             dy = img["dy"].as<int>();
00176         if(dy < 0) dy = -1;
00177         else dy = 1;
00178
00179         int imgw = image.width;
00180         int imgh = image.height;
00181
00182         if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183         {
00184             for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185             {
00186                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                 {
00188                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                     Image img2 = ImageFromImage(image, rect);
00190                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                     sprites.back().push_back(vis);
00193
00194                     UnloadImage(img2);
00195                 }
00196             }
00197         }else
00198         {
00199             for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200             {
00201                 for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202                 {
00203                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204                     Image img2 = ImageFromImage(image, rect);
00205                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207                     sprites.back().push_back(vis);
00208
00209                     UnloadImage(img2);
00210                 }
00211             }
00212         }
00213     }
00214     UnloadImage(image);
00215 }
00216 }

```

### 7.3.3.37 moveBy() [1/2]

```

void Frame::moveBy (
    fPoint rel ) [inherited]

```

Definition at line 65 of file [arithmetic.cpp](#).

```
00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }
```

### 7.3.3.38 moveBy() [2/2]

```
void Frame::moveBy (
    int x,
    int y ) [inherited]
```

Definition at line 75 of file [arithmetic.cpp](#).

```
00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }
```

### 7.3.3.39 moveCenterTo() [1/2]

```
void Frame::moveCenterTo (
    fPoint rel ) [inherited]
```

Definition at line 43 of file [arithmetic.cpp](#).

```
00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }
```

### 7.3.3.40 moveCenterTo() [2/2]

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

**7.3.3.41 moveTo() [1/2]**

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

**7.3.3.42 moveTo() [2/2]**

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.3.3.43 nextImage()**

```
void Container::nextImage ( ) [inherited]
```

move to next state of the sprite

Definition at line 247 of file [constructor.cpp](#).

```
00248 {
00249     if(sprites.empty()) return;
00250     focus[1]++;
00251     if(focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
```

**7.3.3.44 nextSprite()**

```
void Container::nextSprite ( ) [inherited]
```

move to the next sprite

Definition at line 261 of file [constructor.cpp](#).

```
00262 {
00263     if(sprites.empty()) return;
00264     focus[0]++;
00265     if(focus[0] >= sprites.size()) focus[0] = 0;
00266 }
```

**7.3.3.45 operator fRect()**

```
Frame::operator fRect ( ) const [inherited]
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

**7.3.3.46 operator iRect()**

```
Frame::operator iRect ( ) const [inherited]
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

**7.3.3.47 operator Rectangle()**

```
Frame::operator Rectangle ( ) const [inherited]
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

**7.3.3.48 OverlappingArea() [1/2]**

```
float Container::OverlappingArea (
    Container * container ) [inherited]
```

Definition at line 34 of file [overlap.cpp](#).

```
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if(w < 0 || h < 0) return 0;
00043     return w * h;
00044 }
```

**7.3.3.49 OverlappingArea() [2/2]**

```
float Container::OverlappingArea (
    Rectangle rec ) [inherited]
```

Definition at line 23 of file [overlap.cpp](#).

```
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
```

**7.3.3.50 plug() [1/2]**

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position



## Parameters

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```

00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
```

## 7.3.3.51 plug() [2/2]

```

void Frame::plug (
    Frame * par,
    fRect rel ) [inherited]
```

attach a frame to a parent by relative position

## Parameters

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 12 of file [family.cpp](#).

```

00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }
```

## 7.3.3.52 prevImage()

```

void Container::prevImage ( ) [inherited]
```

move to previous state of the sprite

Definition at line 254 of file [constructor.cpp](#).

```

00255 {
00256     if(sprites.empty()) return;
00257     focus[1]--;
00258     if(focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }
```

### 7.3.3.53 prevSprite()

```
void Container::prevSprite ( ) [inherited]
```

move to the previous sprite

Definition at line 268 of file [constructor.cpp](#).

```
00269 {
00270     if(sprites.empty()) return;
00271     focus[0]--;
00272     if(focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }
```

### 7.3.3.54 react()

```
PacketAction * ButtonImage::react ( ) [virtual]
```

Reimplemented from [Container](#).

Definition at line 12 of file [arithmetic.cpp](#).

```
00012 {
00013
00014     if (CheckCollisionPointRec(GetMousePosition(), rectangle)) {
00015         this->isHover = 1;
00016         if (IsMouseButtonDown(MOUSE_LEFT_BUTTON)) { // click -> pressing
00017             this->clicked = true;
00018             if(this->pressingID == -1)
00019                 return nullptr;
00020             PacketAction* packet = new PacketAction();
00021             packet->addAction(actions[pressingID]->clone());
00022             return packet;
00023         }
00024         else if(this->clicked) { // release -> click
00025
00026             this->clicked = false;
00027             if(this->clickedID == -1)
00028                 return nullptr;
00029             PacketAction* packet = new PacketAction();
00030             packet->addAction(actions[clickedID]->clone());
00031             packet->addAction(new changeInfRequest("test"));
00032             return packet;
00033         }
00034         if(this->hoverID == -1)
00035             return nullptr;
00036         PacketAction* packet = new PacketAction();
00037         packet->addAction(actions[hoverID]->clone());
00038         return packet;
00039     }
00040     if (this->isHover == 1)
00041     {
00042         this->isHover = 0;
00043         if(this->releaseID == -1)
00044             return nullptr;
00045         PacketAction* packet = new PacketAction();
00046         packet->addAction(actions[releaseID]->clone());
00047         return packet;
00048     }
00049     return nullptr;
00050 }
```

### 7.3.3.55 removeSubframe()

```
void Frame::removeSubframe (
    Frame * subframe ) [protected], [inherited]
```

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

## Parameters

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

### 7.3.3.56 `resize()` [1/2]

```
void Frame::resize (
    fPoint rel ) [inherited]
```

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

### 7.3.3.57 `resize()` [2/2]

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

### 7.3.3.58 `setProbability()`

```
void Container::setProbability (
    int prob ) [inherited]
```

Definition at line 280 of file [constructor.cpp](#).

```
00281 {
00282     probability = prob;
00283 }
```

### 7.3.3.59 setRelative()

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

### 7.3.3.60 show()

```
void Container::show ( ) [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     visible = true;
00014 }
```

### 7.3.3.61 toggleVisibility()

```
void Container::toggleVisibility ( ) [inherited]
```

Definition at line 21 of file [arithmetic.cpp](#).

```
00022 {
00023     visible = !visible;
00024 }
```

### 7.3.3.62 unplug()

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

### 7.3.3.63 updateFrame()

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual], [inherited]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
```

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

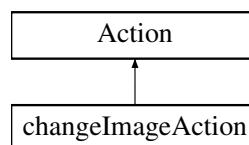
- [src/button/include/button.hpp](#)
- [src/button/src/arithmetic.cpp](#)
- [src/button/src/constructor.cpp](#)
- [src/button/src/destructor.cpp](#)

## 7.4 changelImageAction Class Reference

changes display image of container

```
#include <container.hpp>
```

Inheritance diagram for changelImageAction:



### Public Member Functions

- [changelImageAction](#) ([Container](#) \*, [iPoint](#))
- [changelImageAction](#) ([changelImageAction](#) \*)
- [~changelImageAction](#) ()
- void [execute](#) () override
- [Action](#) \* [clone](#) () override
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.4.1 Detailed Description

changes display image of container

Definition at line 108 of file [container.hpp](#).

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 changeImageAction() [1/2]

```
changeImageAction::changeImageAction (
    Container * c,
    iPoint p )
```

Definition at line 3 of file [changesprite.cpp](#).

```
00004 {
00005     container = c;
00006     focus = p;
00007 }
```

#### 7.4.2.2 changeImageAction() [2/2]

```
changeImageAction::changeImageAction (
    changeImageAction * c )
```

Definition at line 9 of file [changesprite.cpp](#).

```
00010 {
00011     container = c->container;
00012     focus = c->focus;
00013 }
```

#### 7.4.2.3 ~changeImageAction()

```
changeImageAction::~changeImageAction ( )
```

Definition at line 15 of file [changesprite.cpp](#).

```
00016 {
00017     container = nullptr;
00018 }
```

### 7.4.3 Member Function Documentation

#### 7.4.3.1 clone()

```
Action * changeImageAction::clone ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 25 of file [changesprite.cpp](#).

```
00026 {
00027     return new changeImageAction(this);
00028 }
```

#### 7.4.3.2 execute()

```
void changeImageAction::execute ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 20 of file [changesprite.cpp](#).

```
00021 {  
00022     container->chooseImage(focus[0], focus[1]);  
00023 }
```

#### 7.4.3.3 getArgs()

```
ARGS & Action::getArgs ( ) [virtual], [inherited]
```

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

#### 7.4.3.4 isPackage()

```
bool Action::isPackage ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

#### 7.4.3.5 isRequest()

```
int Action::isRequest ( ) [virtual], [inherited]
```

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {  
00017     return 0;  
00018 }
```

#### 7.4.3.6 unpack()

```
std::vector< Action * > Action::unpack ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

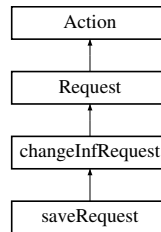
The documentation for this class was generated from the following files:

- [src/container/include/container.hpp](#)
- [src/container/src/action/changesprite.cpp](#)

## 7.5 changeInfRequest Class Reference

```
#include <request.hpp>
```

Inheritance diagram for changeInfRequest:



### Public Member Functions

- [changeInfRequest](#) (std::string s)
- [changeInfRequest](#) ([changeInfRequest](#) \*)
- [~changeInfRequest](#) ()=default
- [int isRequest](#) () override
- [Action \\* clone](#) () override
- [ARGS & getArgs](#) () override
- virtual bool [isPackage](#) ()
- virtual void [execute](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()

### Protected Attributes

- [ARGS args](#)

### 7.5.1 Detailed Description

Definition at line 27 of file [request.hpp](#).

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 changeInfRequest() [1/2]

```
changeInfRequest::changeInfRequest (
    std::string s )
```

Definition at line 3 of file [changeinf.cpp](#).

```
00004 {
00005     args.str.push_back(s);
00006 }
```



### 7.5.2.2 changeInfRequest() [2/2]

```
changeInfRequest::changeInfRequest (
    changeInfRequest * other )
```

Definition at line 8 of file [changeinf.cpp](#).

```
00009 {
00010     args = other->args;
00011 }
```

### 7.5.2.3 ~changeInfRequest()

```
changeInfRequest::~~changeInfRequest ( ) [default]
```

## 7.5.3 Member Function Documentation

### 7.5.3.1 clone()

```
Action * changeInfRequest::clone ( ) [override], [virtual]
```

Reimplemented from [Request](#).

Definition at line 18 of file [changeinf.cpp](#).

```
00019 {
00020     return new changeInfRequest(this);
00021 }
```

### 7.5.3.2 execute()

```
void Action::execute ( ) [virtual], [inherited]
```

Reimplemented in [CloseAction](#), [resizeAction](#), [PacketAction](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 25 of file [action.cpp](#).

```
00026 {
00027 }
```

### 7.5.3.3 getArgs()

```
ARGS & changeInfRequest::getArgs ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 23 of file [changeinf.cpp](#).

```
00024 {
00025     return args;
00026 }
```

### 7.5.3.4 isPackage()

```
bool Action::isPackage ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.5.3.5 isRequest()

```
int changeInfRequest::isRequest ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 13 of file [changeinf.cpp](#).

```
00014 {  
00015     return REQUEST::CHANGE_INF;  
00016 }
```

### 7.5.3.6 unpack()

```
std::vector< Action * > Action::unpack ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

## 7.5.4 Member Data Documentation

### 7.5.4.1 args

```
ARGS Request::args [protected], [inherited]
```

Definition at line 17 of file [request.hpp](#).

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

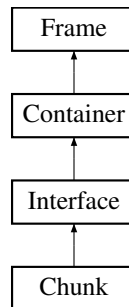
- [src/action/include/request.hpp](#)
- [src/action/src/request/changeinf.cpp](#)

## 7.6 Chunk Class Reference

manages the spawning of chunks and how entities interact with them

```
#include <chunk.hpp>
```

Inheritance diagram for Chunk:



### Public Member Functions

- [Chunk](#) ([Frame](#) \*, [Rectangle](#))
- [Chunk](#) ([Chunk](#) \*)
- [Chunk](#) ([Chunk](#) \*, [Rectangle](#))
- [Chunk](#) ([Chunk](#) \*, [Frame](#) \*, [Rectangle](#))
- [~Chunk](#) ()
- void [addVisitor](#) ([Container](#) \*)
- void [addVisitor](#) ([Container](#) \*, int)
- void [addVisitor](#) ([Container](#) \*, [Rectangle](#))
- void [addVisitor](#) ([Container](#) \*, int, [Rectangle](#))
- void [generateEntity](#) ()
- void [setVelocity](#) ([fPoint](#))
- std::string [linkContent](#) (std::string path) override
- [Action](#) \* [getRuntimeEvent](#) () override
- void [draw](#) () override
- [Container](#) \* [getContainers](#) (int)
- int [getContainersSize](#) ()
- std::string [linkContentAbsolute](#) (std::string path) override
- [Action](#) \* [react](#) () override
- std::string [getName](#) ()
- void [setProbability](#) (int)
- int [getProbability](#) ()
- void [chooseSprite](#) (int)
  - choose a specific sprite from a vector of sprites*
- void [chooseImage](#) (int)
  - choose the state of the sprite*
- void [chooseImage](#) (int, int)
  - choose the state of the sprite*
- void [nextImage](#) ()
  - move to next state of the sprite*
- void [prevImage](#) ()
  - move to previous state of the sprite*
- void [nextSprite](#) ()

- *move to the next sprite*
- void `prevSprite` ()
- *move to the previous sprite*
- bool `isOverlapping` (fPoint)
- bool `isOverlapping` (Rectangle)
- bool `isOverlapping` (Container \*)
- float `OverlappingArea` (Rectangle)
- float `OverlappingArea` (Container \*)
- void `show` ()
- void `hide` ()
- void `toggleVisibility` ()
- bool `isVisible` ()
- int `getInstancelid` ()
- void `plug` (Frame \*par, fRect rel)
- *attach a frame to a parent by relative position*
- void `plug` (Frame \*par)
- *attach a frame to a parent by old relative position*
- void `unplug` ()
- *detach a frame from its parent*
- void `moveTo` (fPoint rel)
- void `moveTo` (int x, int y)
- void `moveCenterTo` (fPoint rel)
- void `moveCenterTo` (int x, int y)
- void `moveBy` (fPoint rel)
- void `moveBy` (int, int)
- void `resize` (fPoint rel)
- void `resize` (int w, int h)
- const Rectangle & `getFrame` () const
- const fRect & `getRelative` () const
- Frame \* `getParent` ()
- void `setRelative` (fRect rel)
- const fPoint & `getCenter` () const
- const float & `getX` () const
- const float & `getY` () const
- const float & `getW` () const
- const float & `getH` () const
- operator Rectangle () const
- operator fRect () const
- operator iRect () const

### Protected Member Functions

- void `drawEntity` ()
- Container \* `randomEntity` ()
- void `movingEntity` ()
- void `loadObject` (YAML::Node)
- void `loadControl` (YAML::Node)
- void `loadButton` (YAML::Node)
- void `drawNested` ()
- void `drawContainers` ()
- bool `loadName` (YAML::Node node)
- void `loadSprites` (YAML::Node node)
- void `loadFocus` (YAML::Node node)

- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) ([Frame](#) \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) ([Frame](#) \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

## Friends

- class [moveEntityAction](#)

## 7.6.1 Detailed Description

manages the spawning of chunks and how entities interact with them

Definition at line 16 of file [chunk.hpp](#).

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 [Chunk\(\)](#) [1/4]

```
Chunk::Chunk (
    Frame * frame,
    Rectangle rect )
```

Definition at line 4 of file [constructor.cpp](#).

```
00004                                     : Interface(frame, rect)
00005 {
00006
00007 }
```

### 7.6.2.2 [Chunk\(\)](#) [2/4]

```
Chunk::Chunk (
    Chunk * other )
```

Definition at line 9 of file [constructor.cpp](#).

```
00009                                     : Interface(other)
00010 {
00011
00012     for(auto i : other->visiter)
00013     {
00014         Rectangle rel;
00015         rel.x = 1;
00016         rel.y = -0.375;
00017         rel.width = i->getRelative()[2];
00018         rel.height = i->getRelative()[3];
00019         visiter.push_back(new Container(i, this, rel));
00020     }
00021     velocity = other->velocity;
00022     generateEntity();
00023 }
```

### 7.6.2.3 Chunk() [3/4]

```
Chunk::Chunk (
    Chunk * other,
    Rectangle rect )
```

Definition at line 25 of file [constructor.cpp](#).

```
00025                                     : Interface(other, rect)
00026 {
00027     for(auto i : other->visiter)
00028     {
00029         Rectangle rel;
00030         rel.x = 1;
00031         rel.y = -0.375;
00032         rel.width = i->getRelative()[2];
00033         rel.height = i->getRelative()[3];
00034         visiter.push_back(new Container(i, this, rel));
00035     }
00036     velocity = other->velocity;
00037     generateEntity();
00038 }
```

### 7.6.2.4 Chunk() [4/4]

```
Chunk::Chunk (
    Chunk * other,
    Frame * frame,
    Rectangle rect )
```

Definition at line 40 of file [constructor.cpp](#).

```
00040                                     : Interface(other, frame, rect)
00041 {
00042     for(auto i : other->visiter)
00043     {
00044         Rectangle rel;
00045         rel.x = 1;
00046         rel.y = -0.375;
00047         rel.width = i->getRelative()[2];
00048         rel.height = i->getRelative()[3];
00049         visiter.push_back(new Container(i, this, rel));
00050     }
00051     velocity = other->velocity;
00052     generateEntity();
00053 }
```

### 7.6.2.5 ~Chunk()

```
Chunk::~Chunk ( )
```

Definition at line 3 of file [destructor.cpp](#).

```
00004 {
00005     for(auto i : visiter)
00006         delete i;
00007
00008     while(!Entity.empty())
00009     {
00010         delete Entity.back();
00011         Entity.pop_back();
00012     }
00013 }
```

## 7.6.3 Member Function Documentation

### 7.6.3.1 addSubframe()

```
void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]
```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

## Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```
00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }
```

### 7.6.3.2 addVisiter() [1/4]

```
void Chunk::addVisiter (
    Container * obj )
```

Definition at line 82 of file [constructor.cpp](#).

```
00083 {
00084     Rectangle rel;
00085     rel.x = obj->getRelative()[0];
00086     rel.y = obj->getRelative()[1];
00087     rel.width = obj->getRelative()[2];
00088     rel.height = obj->getRelative()[3];
00089
00090     Container* c = new Container(obj, this, rel);
00091     visiter.push_back(c);
00092 }
```

### 7.6.3.3 addVisiter() [2/4]

```
void Chunk::addVisiter (
    Container * obj,
    int prob )
```

Definition at line 94 of file [constructor.cpp](#).

```
00095 {
00096     Rectangle rel;
00097     rel.x = obj->getRelative()[0];
00098     rel.y = obj->getRelative()[1];
00099     rel.width = obj->getRelative()[2];
00100     rel.height = obj->getRelative()[3];
00101
00102     Container* c = new Container(obj, this, rel);
00103     c->setProbability(prob);
00104     visiter.push_back(c);
00105 }
```

### 7.6.3.4 addVisiter() [3/4]

```
void Chunk::addVisiter (
    Container * obj,
    int prob,
    Rectangle rel )
```

Definition at line 113 of file [constructor.cpp](#).

```
00114 {
00115     Container* c = new Container(obj, this, rel);
00116     c->setProbability(prob);
00117     visiter.push_back(c);
00118 }
```

**7.6.3.5 addVisiter() [4/4]**

```
void Chunk::addVisiter (
    Container * obj,
    Rectangle rel )
```

Definition at line 107 of file [constructor.cpp](#).

```
00108 {
00109     Container* c = new Container(obj, this, rel);
00110     visiter.push_back(c);
00111 }
```

**7.6.3.6 beginUpdate()**

```
void Frame::beginUpdate ( ) [protected], [inherited]
```

Definition at line 113 of file [family.cpp](#).

```
00114 {
00115     mtx.lock();
00116 }
```

**7.6.3.7 chooseImage() [1/2]**

```
void Container::chooseImage (
    int index ) [inherited]
```

choose the state of the sprite

Definition at line 231 of file [constructor.cpp](#).

```
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
```

**7.6.3.8 chooseImage() [2/2]**

```
void Container::chooseImage (
    int index,
    int index2 ) [inherited]
```

choose the state of the sprite

Definition at line 238 of file [constructor.cpp](#).

```
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
```

**7.6.3.9 chooseSprite()**

```
void Container::chooseSprite (
    int index ) [inherited]
```

choose a specific sprite from a vector of sprites

Definition at line 224 of file [constructor.cpp](#).

```
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
```



#### 7.6.3.10 draw()

```
void Chunk::draw ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 9 of file [arithmetic.cpp](#).

```
00010 {  
00011  
00012     Container::draw();  
00013     drawNested();  
00014     drawContainers();  
00015     drawEntity();  
00016 }
```

#### 7.6.3.11 drawContainers()

```
void Interface::drawContainers ( ) [protected], [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {  
00013     for(auto& child : containers)  
00014     {  
00015         child->draw();  
00016     }  
00017 }
```

#### 7.6.3.12 drawEntity()

```
void Chunk::drawEntity ( ) [protected]
```

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {  
00005     for(auto i : Entity)  
00006         i->draw();  
00007 }
```

#### 7.6.3.13 drawNested()

```
void Interface::drawNested ( ) [protected], [inherited]
```

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {  
00005     for(auto& child : nested)  
00006     {  
00007         child->draw();  
00008     }  
00009 }
```

#### 7.6.3.14 endUpdate()

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {  
00120     mtx.unlock();  
00121 }
```

**7.6.3.15 generateEntity()**

```
void Chunk::generateEntity ( )
```

Definition at line 55 of file [constructor.cpp](#).

```
00056 {
00057
00058     if(visiter.empty()) return;
00059     float x = GetRandomValue(-40, 10);
00060
00061     while(x < 0.9)
00062     {
00063         Container* c = randomEntity();
00064         Rectangle rel;
00065         rel.x = x;
00066         rel.y = -0.375;
00067         rel.width = c->getRelative()[2];
00068         rel.height = c->getRelative()[3];
00069         Container* cont = new Container(c, this, rel);
00070         Entity.push_back(cont);
00071         x += GetRandomValue(20, 60) / 100.0;
00072     }
00073 }
```

**7.6.3.16 getCenter()**

```
const fPoint & Frame::getCenter ( ) const [inherited]
```

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139
00140     return resu;
00141 }
```

**7.6.3.17 getContainers()**

```
Container * Interface::getContainers (
    int id ) [inherited]
```

Definition at line 176 of file [constructor.cpp](#).

```
00177 {
00178     if(id < 0 || id >= containers.size()) return nullptr;
00179     return containers[id];
00180 }
```

**7.6.3.18 getContainersSize()**

```
int Interface::getContainersSize ( ) [inherited]
```

Definition at line 182 of file [constructor.cpp](#).

```
00183 {
00184     return containers.size();
00185 }
```

### 7.6.3.19 getFrame()

const Rectangle & Frame::getFrame ( ) const [inherited]

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {  
00107     std::lock_guard<std::mutex> lock(mtx);  
00108     return frame;  
00109 }
```

### 7.6.3.20 getH()

const float & Frame::getH ( ) const [inherited]

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```

### 7.6.3.21 getInstanceId()

int Container::getInstanceId ( ) [inherited]

Definition at line 31 of file [arithmetic.cpp](#).

```
00032 {  
00033     return instance_id;  
00034 }
```

### 7.6.3.22 getName()

std::string Container::getName ( ) [inherited]

Definition at line 275 of file [constructor.cpp](#).

```
00276 {  
00277     return name;  
00278 }
```

### 7.6.3.23 getParent()

Frame \* Frame::getParent ( ) [inherited]

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```

### 7.6.3.24 getProbability()

int Container::getProbability ( ) [inherited]

Definition at line 285 of file [constructor.cpp](#).

```
00286 {  
00287     return probability;  
00288 }
```

**7.6.3.25 getRelative()**

```
const fRect & Frame::getRelative ( ) const [inherited]
```

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {
00113     std::lock_guard<std::mutex> lock(mtx);
00114     return relative;
00115 }
```

**7.6.3.26 getRuntimeEvent()**

```
Action * Chunk::getRuntimeEvent ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 26 of file [action.cpp](#).

```
00027 {
00028     PacketAction* packet = nullptr;
00029     Action* action = Interface::getRuntimeEvent();
00030
00031     if(action != nullptr)
00032     {
00033         packet = new PacketAction();
00034         packet->addAction(action);
00035     }
00036
00037     if(std::chrono::system_clock::now() - moveClock >= moveTime)
00038     {
00039         Action* action = new moveEntityAction(this);
00040         if(packet == nullptr)
00041         {
00042             packet = new PacketAction();
00043         }
00044         packet->addAction(action);
00045         moveClock = std::chrono::system_clock::now();
00046     }
00047     return packet;
00048 }
```

**7.6.3.27 getW()**

```
const float & Frame::getW ( ) const [inherited]
```

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {
00157     std::lock_guard<std::mutex> lock(mtx);
00158     return frame.width;
00159 }
```

**7.6.3.28 getX()**

```
const float & Frame::getX ( ) const [inherited]
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {
00145     std::lock_guard<std::mutex> lock(mtx);
00146     return frame.x;
00147 }
```

### 7.6.3.29 getY()

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {
00151     std::lock_guard<std::mutex> lock (mtx);
00152     return frame.y;
00153 }
```

### 7.6.3.30 hide()

```
void Container::hide ( ) [inherited]
```

Definition at line 16 of file [arithmetic.cpp](#).

```
00017 {
00018     visible = false;
00019 }
```

### 7.6.3.31 isOverlapping() [1/3]

```
bool Container::isOverlapping (
    Container * container ) [inherited]
```

Definition at line 16 of file [overlap.cpp](#).

```
00017 {
00018     Rectangle rec = getFrame();
00019     Rectangle rec2 = container->getFrame();
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
        rec2.height && rec.y + rec.height >= rec2.y);
00021 }
```

### 7.6.3.32 isOverlapping() [2/3]

```
bool Container::isOverlapping (
    fPoint point ) [inherited]
```

Definition at line 3 of file [overlap.cpp](#).

```
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
        rec.y + rec.height);
00007
00008 }
```

### 7.6.3.33 isOverlapping() [3/3]

```
bool Container::isOverlapping (
    Rectangle rec ) [inherited]
```

Definition at line 10 of file [overlap.cpp](#).

```
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
        rec2.height && rec.y + rec.height >= rec2.y);
00014 }
```

### 7.6.3.34 isroot()

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

### 7.6.3.35 isVisible()

```
bool Container::isVisible ( ) [inherited]
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     return visible;
00029 }
```

### 7.6.3.36 linkContent()

```
std::string Chunk::linkContent (
    std::string path ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 76 of file [constructor.cpp](#).

```
00077 {
00078     return linkContentAbsolute(PATB::CHUNK_ + path);
00079 }
```

### 7.6.3.37 linkContentAbsolute()

```
std::string Interface::linkContentAbsolute (
    std::string path ) [override], [virtual], [inherited]
```

Reimplemented from [Container](#).

Definition at line 83 of file [constructor.cpp](#).

```
00084 {
00085     YAML::Node node = YAML_FILE::readFile(path);
00086     if(!loadName(node)) return "";
00087
00088     if(node["textures"])
00089         loadSprites(node["textures"]);
00090
00091     if(node["focus"])
00092         loadFocus(node["focus"]);
00093     else chooseImage(0, 0);
00094
00095     if(node["object"])
00096         loadObject(node["object"]);
00097
00098     if(node["control"])
00099         loadControl(node["control"]);
00100
00101     if(node["button"])
00102         loadButton(node["button"]);
00103
00104     // if(node["collide"])
00105     //     loadCollide(node["collide"]);
00106
00107     // if(node["chunk"])
00108     //     loadChunk(node["chunk"]);
00109
00110
00111     // if(node["event"])
00112     //     loadEvent(node["event"]);
00113
00114     return getName();
00115 }
```

### 7.6.3.38 loadButton()

```
void Interface::loadButton (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 159 of file [constructor.cpp](#).

```
00160 {
00161     for(auto i : node)
00162     {
00163         Rectangle rel({0, 0, 0, 0});
00164         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00165         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00166         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00167         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00168         ButtonImage *obj;
00169         obj = new ButtonImage(this, rel);
00170         obj->linkContent(i["path"].as<std::string>());
00171         obj->show();
00172         containers.push_back(obj);
00173     }
00174 }
```

### 7.6.3.39 loadControl()

```
void Interface::loadControl (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 134 of file [constructor.cpp](#).

```
00135 {
00136     for(auto stroke : node)
00137     {
00138         KeyStroke* k = new KeyStroke();
00139         for(auto key : stroke["key"])
00140         {
00141             k->add(toKey(key.as<std::string>()));
00142         }
00143         std::string action = stroke["action"].as<std::string>();
00144
00145         if(action == "move-object")
00146         {
00147             int id = stroke["args"][0].as<int>();
00148             float v = stroke["args"][1].as<float>() / 100.0;
00149             float x = stroke["args"][2].as<float>();
00150             float y = stroke["args"][3].as<float>();
00151             moveObjectAction* action = new moveObjectAction(containers[id], fPoint({x, y}), v);
00152             k->addAction(action);
00153         }
00154         keystrokes.push_back(k);
00155     }
00156 }
00157 }
```

### 7.6.3.40 loadFocus()

```
void Container::loadFocus (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 218 of file [constructor.cpp](#).

```
00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }
```

### 7.6.3.41 loadName()

```
bool Container::loadName (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 111 of file [constructor.cpp](#).

```
00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }
```

### 7.6.3.42 loadObject()

```
void Interface::loadObject (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 117 of file [constructor.cpp](#).

```
00118 {
00119     for(auto i : node)
00120     {
00121         Rectangle rel({0, 0, 0, 0});
00122         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00123         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00124         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00125         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00126         Container *obj;
00127         obj = new Object(this, rel);
00128         obj->linkContent(i["path"].as<std::string>());
00129         containers.push_back(obj);
00130     }
00131 }
```

### 7.6.3.43 loadSprites()

```
void Container::loadSprites (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 122 of file [constructor.cpp](#).

```
00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])
00141         {
00142             float x, y, w, h;
00143             int repeat = 1;
00144             int gapX = 0;
00145             int gapY = 0;
00146
00147             int dx = 1;
00148             int dy = 1;
00149
00150             if(img["x"])
00151                 x = img["x"].as<float>() / 100.0;
```



```

00152         else x = 0;
00153         if(img["y"])
00154             y = img["y"].as<float>() / 100.0;
00155         else y = 0;
00156         if(img["w"])
00157             w = img["w"].as<float>() / 100.0;
00158         else w = 1;
00159         if(img["h"])
00160             h = img["h"].as<float>() / 100.0;
00161         else h = 1;
00162         if(img["repeat"])
00163             repeat = img["repeat"].as<int>();
00164         if(img["gapX"])
00165             gapX = img["gapX"].as<int>();
00166         if(img["gapY"])
00167             gapY = img["gapY"].as<int>();
00168
00169         if(img["dx"])
00170             dx = img["dx"].as<int>();
00171         if(dx < 0) dx = -1;
00172         else dx = 1;
00173
00174         if(img["dy"])
00175             dy = img["dy"].as<int>();
00176         if(dy < 0) dy = -1;
00177         else dy = 1;
00178
00179         int imgw = image.width;
00180         int imgh = image.height;
00181
00182         if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183         {
00184             for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185             {
00186                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                 {
00188                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                     Image img2 = ImageFromImage(image, rect);
00190                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                     sprites.back().push_back(vis);
00193
00194                     UnloadImage(img2);
00195                 }
00196             }
00197         }else
00198         {
00199             for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200             {
00201                 for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202                 {
00203                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204                     Image img2 = ImageFromImage(image, rect);
00205                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207                     sprites.back().push_back(vis);
00208
00209                     UnloadImage(img2);
00210                 }
00211             }
00212         }
00213     }
00214     UnloadImage(image);
00215 }
00216 }

```

### 7.6.3.44 moveBy() [1/2]

```

void Frame::moveBy (
    fPoint rel ) [inherited]

```

Definition at line 65 of file [arithmetic.cpp](#).

```

00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }

```

### 7.6.3.45 moveBy() [2/2]

```
void Frame::moveBy (
    int x,
    int y ) [inherited]
```

Definition at line 75 of file [arithmetic.cpp](#).

```
00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }
```

### 7.6.3.46 moveCenterTo() [1/2]

```
void Frame::moveCenterTo (
    fPoint rel ) [inherited]
```

Definition at line 43 of file [arithmetic.cpp](#).

```
00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }
```

### 7.6.3.47 moveCenterTo() [2/2]

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

### 7.6.3.48 moveTo() [1/2]

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

**7.6.3.49 moveTo() [2/2]**

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.6.3.50 movingEntity()**

```
void Chunk::movingEntity ( ) [protected]
```

Definition at line 17 of file [action.cpp](#).

```
00018 {
00019     for(auto i : Entity)
00020     {
00021         i->moveBy(velocity);
00022         i->nextImage();
00023     }
00024 }
```

**7.6.3.51 nextImage()**

```
void Container::nextImage ( ) [inherited]
```

move to next state of the sprite

Definition at line 247 of file [constructor.cpp](#).

```
00248 {
00249     if(sprites.empty()) return;
00250     focus[1]++;
00251     if(focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
```

**7.6.3.52 nextSprite()**

```
void Container::nextSprite ( ) [inherited]
```

move to the next sprite

Definition at line 261 of file [constructor.cpp](#).

```
00262 {
00263     if(sprites.empty()) return;
00264     focus[0]++;
00265     if(focus[0] >= sprites.size()) focus[0] = 0;
00266 }
```

**7.6.3.53 operator fRect()**

```
Frame::operator fRect ( ) const [inherited]
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

**7.6.3.54 operator iRect()**

```
Frame::operator iRect ( ) const [inherited]
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

**7.6.3.55 operator Rectangle()**

```
Frame::operator Rectangle ( ) const [inherited]
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

**7.6.3.56 OverlappingArea() [1/2]**

```
float Container::OverlappingArea (
    Container * container ) [inherited]
```

Definition at line 34 of file [overlap.cpp](#).

```
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if(w < 0 || h < 0) return 0;
00043     return w * h;
00044 }
```

**7.6.3.57 OverlappingArea() [2/2]**

```
float Container::OverlappingArea (
    Rectangle rec ) [inherited]
```

Definition at line 23 of file [overlap.cpp](#).

```
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
```

**7.6.3.58 plug() [1/2]**

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position

## Parameters

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```

00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
```

## 7.6.3.59 plug() [2/2]

```

void Frame::plug (
    Frame * par,
    fRect rel ) [inherited]
```

attach a frame to a parent by relative position

## Parameters

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 12 of file [family.cpp](#).

```

00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }
```

## 7.6.3.60 prevImage()

```

void Container::prevImage ( ) [inherited]
```

move to previous state of the sprite

Definition at line 254 of file [constructor.cpp](#).

```

00255 {
00256     if(sprites.empty()) return;
00257     focus[1]--;
00258     if(focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }
```

**7.6.3.61 prevSprite()**

```
void Container::prevSprite ( ) [inherited]
```

move to the previous sprite

Definition at line 268 of file [constructor.cpp](#).

```
00269 {
00270     if(sprites.empty()) return;
00271     focus[0]--;
00272     if(focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }
```

**7.6.3.62 randomEntity()**

```
Container * Chunk::randomEntity ( ) [protected]
```

Definition at line 4 of file [action.cpp](#).

```
00005 {
00006     int value = GetRandomValue(0, 100);
00007
00008     for(auto i : visiter)
00009     {
00010         value -= i->getProbability();
00011         if(value <= 0) return i;
00012     }
00013
00014     return visiter[GetRandomValue(0, visiter.size() - 1)];
00015 }
```

**7.6.3.63 react()**

```
Action * Interface::react ( ) [override], [virtual], [inherited]
```

Reimplemented from [Container](#).

Definition at line 38 of file [action.cpp](#).

```
00039 {
00040     if(!isVisible()) return nullptr;
00041     PacketAction* packet = nullptr;
00042
00043     Action* action = Container::react();
00044
00045     if(action != nullptr)
00046     {
00047         packet = new PacketAction();
00048         packet->addAction(action);
00049     }
00050
00051     for(auto i : keystrokes)
00052     {
00053         Action* action = i->react();
00054         if(action != nullptr)
00055         {
00056             if(packet == nullptr) packet = new PacketAction();
00057             packet->addAction(action);
00058         }
00059     }
00060
00061     for(auto i : containers)
00062     {
00063         Action* action = i->react();
00064         if(action != nullptr)
00065         {
00066             if(packet == nullptr) packet = new PacketAction();
00067             packet->addAction(action);
00068         }
00069     }
00070
00071     return packet;
00072 }
```

### 7.6.3.64 removeSubframe()

```
void Frame::removeSubframe (
    Frame * subframe ) [protected], [inherited]
```

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

### 7.6.3.65 resize() [1/2]

```
void Frame::resize (
    fPoint rel ) [inherited]
```

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

### 7.6.3.66 resize() [2/2]

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

### 7.6.3.67 setProbability()

```
void Container::setProbability (
    int prob ) [inherited]
```

Definition at line 280 of file [constructor.cpp](#).

```
00281 {
00282     probability = prob;
00283 }
```

### 7.6.3.68 setRelative()

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

### 7.6.3.69 setVelocity()

```
void Chunk::setVelocity (
    fPoint vel )
```

Definition at line 120 of file [constructor.cpp](#).

```
00121 {
00122     velocity = vel;
00123 }
```

### 7.6.3.70 show()

```
void Container::show ( ) [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     visible = true;
00014 }
```

### 7.6.3.71 toggleVisibility()

```
void Container::toggleVisibility ( ) [inherited]
```

Definition at line 21 of file [arithmetic.cpp](#).

```
00022 {
00023     visible = !visible;
00024 }
```



### 7.6.3.72 unplug()

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

### 7.6.3.73 updateFrame()

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual], [inherited]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
```

## 7.6.4 Friends And Related Symbol Documentation

### 7.6.4.1 moveEntityAction

```
friend class moveEntityAction [friend]
```

Definition at line 19 of file [chunk.hpp](#).

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

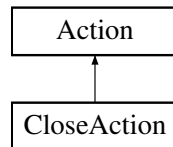
- [src/chunk/include/chunk.hpp](#)
- [src/chunk/src/action.cpp](#)
- [src/chunk/src/arithmetic.cpp](#)
- [src/chunk/src/constructor.cpp](#)
- [src/chunk/src/destructor.cpp](#)

## 7.7 CloseAction Class Reference

manages the closing of the application

```
#include <window.hpp>
```

Inheritance diagram for CloseAction:



### Public Member Functions

- [CloseAction](#) ([Window](#) \*win)
- [~CloseAction](#) ()=default
- void [execute](#) ()
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual [Action](#) \* [clone](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.7.1 Detailed Description

manages the closing of the application

Definition at line [181](#) of file [window.hpp](#).

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 CloseAction()

```
CloseAction::CloseAction (
    Window * win )
```

Definition at line [3](#) of file [close.cpp](#).

```
00004 {
00005     win = window;
00006 }
```

#### 7.7.2.2 ~CloseAction()

```
CloseAction::~~CloseAction ( ) [default]
```

## 7.7.3 Member Function Documentation

### 7.7.3.1 clone()

`Action * Action::clone ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#), [Request](#), [changeInfRequest](#), [loseRequest](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 29 of file [action.cpp](#).

```
00030 {  
00031     return this;  
00032 }
```

### 7.7.3.2 execute()

`void CloseAction::execute ( ) [virtual]`

Reimplemented from [Action](#).

Definition at line 8 of file [close.cpp](#).

```
00009 {  
00010     win->Wcontent.setStatus(false);  
00011 }
```

### 7.7.3.3 getArgs()

`ARGS & Action::getArgs ( ) [virtual], [inherited]`

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.7.3.4 isPackage()

`bool Action::isPackage ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.7.3.5 isRequest()

`int Action::isRequest ( ) [virtual], [inherited]`

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {  
00017     return 0;  
00018 }
```

### 7.7.3.6 unpack()

```
std::vector< Action * > Action::unpack ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {
00036     return std::vector<Action*> ({this});
00037 }
```

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

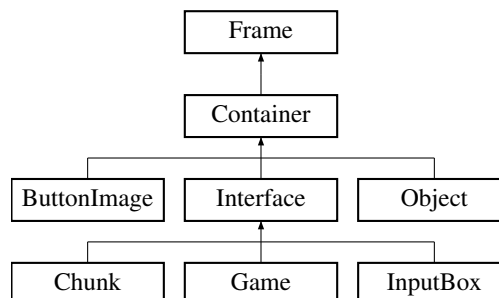
- [src/window/include/window.hpp](#)
- [src/window/src/action/close.cpp](#)

## 7.8 Container Class Reference

holds specific entities and their behavior

```
#include <container.hpp>
```

Inheritance diagram for Container:



### Public Member Functions

- [Container](#) ([Frame](#) \*, [Rectangle](#))
- [Container](#) ([Container](#) \*)
- [Container](#) ([Container](#) \*, [Rectangle](#))
- [Container](#) ([Container](#) \*, [Frame](#) \*, [Rectangle](#))
- virtual [~Container](#) ()
- virtual std::string [linkContent](#) (std::string)
- virtual std::string [linkContentAbsolute](#) (std::string)
- std::string [getName](#) ()
- void [setProbability](#) (int)
- int [getProbability](#) ()
- void [chooseSprite](#) (int)  
    *choose a specific sprite from a vector of sprites*
- void [chooseImage](#) (int)  
    *choose the state of the sprite*
- void [chooseImage](#) (int, int)

- choose the state of the sprite*
- void [nextImage](#) ()
- move to next state of the sprite*
- void [prevImage](#) ()
- move to previous state of the sprite*
- void [nextSprite](#) ()
- move to the next sprite*
- void [prevSprite](#) ()
- move to the previous sprite*
- bool [isOverlapping](#) (fPoint)
- bool [isOverlapping](#) (Rectangle)
- bool [isOverlapping](#) (Container \*)
- float [OverlappingArea](#) (Rectangle)
- float [OverlappingArea](#) (Container \*)
- virtual void [draw](#) ()
- void [show](#) ()
- void [hide](#) ()
- void [toggleVisibility](#) ()
- bool [isVisible](#) ()
- int [getInstanceld](#) ()
- virtual Action \* [react](#) ()
- virtual Action \* [getRuntimeEvent](#) ()
- void [plug](#) (Frame \*par, fRect rel)
  - attach a frame to a parent by relative position*
- void [plug](#) (Frame \*par)
  - attach a frame to a parent by old relative position*
- void [unplug](#) ()
  - detach a frame from its parent*
- void [moveTo](#) (fPoint rel)
- void [moveTo](#) (int x, int y)
- void [moveCenterTo](#) (fPoint rel)
- void [moveCenterTo](#) (int x, int y)
- void [moveBy](#) (fPoint rel)
- void [moveBy](#) (int, int)
- void [resize](#) (fPoint rel)
- void [resize](#) (int w, int h)
- const Rectangle & [getFrame](#) () const
- const fRect & [getRelative](#) () const
- Frame \* [getParent](#) ()
- void [setRelative](#) (fRect rel)
- const fPoint & [getCenter](#) () const
- const float & [getX](#) () const
- const float & [getY](#) () const
- const float & [getW](#) () const
- const float & [getH](#) () const
- operator Rectangle () const
- operator fRect () const
- operator iRect () const

### Protected Member Functions

- bool [loadName](#) (YAML::Node node)
- void [loadSprites](#) (YAML::Node node)
- void [loadFocus](#) (YAML::Node node)
- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) (Frame \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) (Frame \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

### Friends

- class [changeImageAction](#)

## 7.8.1 Detailed Description

holds specific entities and their behavior

Definition at line 19 of file [container.hpp](#).

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 Container() [1/4]

```
Container::Container (
    Frame * parent,
    Rectangle rect )
```

Definition at line 10 of file [constructor.cpp](#).

```
00010 : Frame(parent, rect)
00011 {
00012     instance_id = id_count++;
00013     focus = {0, 0};
00014     visible = true;
00015 }
```

### 7.8.2.2 Container() [2/4]

```
Container::Container (
    Container * other )
```

Definition at line 17 of file [constructor.cpp](#).

```
00017                                     : Frame(other)
00018 {
00019     instance_id = id_count++;
00020     focus = {0, 0};
00021     name = other->name;
00022     visible = true;
00023
00024     for(auto s : other->sprites)
00025     {
00026         sprites.emplace_back();
00027         Rectangle rect;
00028         rect.x = other->getRelative()[0];
00029         rect.y = other->getRelative()[1];
00030         rect.width = other->getRelative()[2];
00031         rect.height = other->getRelative()[3];
00032
00033         for(auto v : s)
00034         {
00035             sprites.back().push_back(new Visual(v, this, rect));
00036         }
00037     }
00038 }
```

### 7.8.2.3 Container() [3/4]

```
Container::Container (
    Container * other,
    Rectangle rect )
```

Definition at line 40 of file [constructor.cpp](#).

```
00040                                     : Frame(other)
00041 {
00042     instance_id = id_count++;
00043     focus = {0, 0};
00044     name = other->name;
00045     setRelative({rect.x, rect.y, rect.width, rect.height});
00046     visible = true;
00047
00048     for(auto s : other->sprites)
00049     {
00050         sprites.emplace_back();
00051         Rectangle rect;
00052         rect.x = other->getRelative()[0];
00053         rect.y = other->getRelative()[1];
00054         rect.width = other->getRelative()[2];
00055         rect.height = other->getRelative()[3];
00056
00057         for(auto v : s)
00058         {
00059             sprites.back().push_back(new Visual(v, this, rect));
00060         }
00061     }
00062 }
```

### 7.8.2.4 Container() [4/4]

```
Container::Container (
    Container * other,
    Frame * parent,
    Rectangle rect )
```

Definition at line 64 of file [constructor.cpp](#).

```
00064                                     : Frame(parent, rect)
00065 {
```

```

00066     instance_id = id_count++;
00067     focus = {0, 0};
00068     name = other->name;
00069     visible = true;
00070     for(auto s : other->sprites)
00071     {
00072         sprites.emplace_back();
00073         Rectangle rect;
00074         rect.x = other->getRelative()[0];
00075         rect.y = other->getRelative()[1];
00076         rect.width = other->getRelative()[2];
00077         rect.height = other->getRelative()[3];
00078
00079         for(auto v : s)
00080         {
00081             sprites.back().push_back(new Visual(v, this, rect));
00082         }
00083     }
00084 }

```

### 7.8.2.5 ~Container()

```
Container::~~Container ( ) [virtual]
```

Definition at line 3 of file [destructor.cpp](#).

```

00004 {
00005     for(Sprite & sprite : sprites)
00006     {
00007         for(auto& frame : sprite)
00008             delete frame;
00009         sprite.clear();
00010     }
00011 }

```

## 7.8.3 Member Function Documentation

### 7.8.3.1 addSubframe()

```
void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]
```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```

00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }

```

### 7.8.3.2 beginUpdate()

```
void Frame::beginUpdate ( ) [protected], [inherited]
```

Definition at line 113 of file [family.cpp](#).

```

00114 {
00115     mtx.lock();
00116 }

```



### 7.8.3.3 chooseImage() [1/2]

```
void Container::chooseImage (
    int index )
```

choose the state of the sprite

Definition at line 231 of file [constructor.cpp](#).

```
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
```

### 7.8.3.4 chooseImage() [2/2]

```
void Container::chooseImage (
    int index,
    int index2 )
```

choose the state of the sprite

Definition at line 238 of file [constructor.cpp](#).

```
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
```

### 7.8.3.5 chooseSprite()

```
void Container::chooseSprite (
    int index )
```

choose a specific sprite from a vector of sprites

Definition at line 224 of file [constructor.cpp](#).

```
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
```

### 7.8.3.6 draw()

```
void Container::draw ( ) [virtual]
```

Reimplemented in [ButtonImage](#), [Chunk](#), [Game](#), [Interface](#), and [Object](#).

Definition at line 4 of file [arithmetic.cpp](#).

```
00005 {
00006     if(sprites.empty()) return;
00007     if(!visible) return;
00008     sprites[focus[0]][focus[1]]->draw();
00009 }
```

### 7.8.3.7 endUpdate()

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {  
00120     mtx.unlock();  
00121 }
```

### 7.8.3.8 getCenter()

```
const fPoint & Frame::getCenter ( ) const [inherited]
```

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {  
00133     std::lock_guard<std::mutex> lock(mtx);  
00134     static fPoint resu;  
00135     if(isroot())  
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};  
00137     else  
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};  
00139  
00140     return resu;  
00141 }
```

### 7.8.3.9 getFrame()

```
const Rectangle & Frame::getFrame ( ) const [inherited]
```

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {  
00107     std::lock_guard<std::mutex> lock(mtx);  
00108     return frame;  
00109 }
```

### 7.8.3.10 getH()

```
const float & Frame::getH ( ) const [inherited]
```

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```

### 7.8.3.11 getInstanceId()

```
int Container::getInstanceId ( )
```

Definition at line 31 of file [arithmetic.cpp](#).

```
00032 {  
00033     return instance_id;  
00034 }
```

### 7.8.3.12 getName()

```
std::string Container::getName ( )
```

Definition at line 275 of file [constructor.cpp](#).

```
00276 {  
00277     return name;  
00278 }
```

### 7.8.3.13 getParent()

```
Frame * Frame::getParent ( ) [inherited]
```

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock (mtx);  
00120     return parent;  
00121 }
```

### 7.8.3.14 getProbability()

```
int Container::getProbability ( )
```

Definition at line 285 of file [constructor.cpp](#).

```
00286 {  
00287     return probability;  
00288 }
```

### 7.8.3.15 getRelative()

```
const fRect & Frame::getRelative ( ) const [inherited]
```

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {  
00113     std::lock_guard<std::mutex> lock (mtx);  
00114     return relative;  
00115 }
```

### 7.8.3.16 getRuntimeEvent()

```
Action * Container::getRuntimeEvent ( ) [virtual]
```

Reimplemented in [Chunk](#), [Game](#), and [Interface](#).

Definition at line 41 of file [arithmetic.cpp](#).

```
00042 {  
00043     return nullptr;  
00044 }
```

### 7.8.3.17 getW()

```
const float & Frame::getW ( ) const [inherited]
```

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {  
00157     std::lock_guard<std::mutex> lock (mtx);  
00158     return frame.width;  
00159 }
```

**7.8.3.18 getX()**

```
const float & Frame::getX ( ) const [inherited]
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {
00145     std::lock_guard<std::mutex> lock(mtx);
00146     return frame.x;
00147 }
```

**7.8.3.19 getY()**

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {
00151     std::lock_guard<std::mutex> lock(mtx);
00152     return frame.y;
00153 }
```

**7.8.3.20 hide()**

```
void Container::hide ( )
```

Definition at line 16 of file [arithmetic.cpp](#).

```
00017 {
00018     visible = false;
00019 }
```

**7.8.3.21 isOverlapping() [1/3]**

```
bool Container::isOverlapping (
    Container * container )
```

Definition at line 16 of file [overlap.cpp](#).

```
00017 {
00018     Rectangle rec = getFrame();
00019     Rectangle rec2 = container->getFrame();
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
    rec2.height && rec.y + rec.height >= rec2.y);
00021 }
```

**7.8.3.22 isOverlapping() [2/3]**

```
bool Container::isOverlapping (
    fPoint point )
```

Definition at line 3 of file [overlap.cpp](#).

```
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
    rec.y + rec.height);
00007
00008 }
```

**7.8.3.23 isOverlapping() [3/3]**

```
bool Container::isOverlapping (
    Rectangle rec )
```

Definition at line 10 of file [overlap.cpp](#).

```
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
        rec2.height && rec.y + rec.height >= rec2.y);
00014 }
```

**7.8.3.24 isroot()**

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

**7.8.3.25 isVisible()**

```
bool Container::isVisible ( )
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     return visible;
00029 }
```

**7.8.3.26 linkContent()**

```
std::string Container::linkContent (
    std::string path ) [virtual]
```

Reimplemented in [Chunk](#), [Interface](#), [ButtonImage](#), and [Object](#).

Definition at line 86 of file [constructor.cpp](#).

```
00087 {
00088     focus = {0, 0};
00089     return linkContentAbsolute(PATB::CONTAINER_ + path);
00090 }
```

**7.8.3.27 linkContentAbsolute()**

```
std::string Container::linkContentAbsolute (
    std::string path ) [virtual]
```

Reimplemented in [Game](#), [Interface](#), [ButtonImage](#), and [Object](#).

Definition at line 92 of file [constructor.cpp](#).

```
00093 {
00094     YAML::Node node = YAML_FILE::readFile(path);
00095     if(!loadName(node)) return "";
00096
00097     if(node["textures"])
00098     {
00099         loadSprites(node["textures"]);
00100     }
00101
00102     if(node["focus"])
00103     {
00104         loadFocus(node["focus"]);
00105     }
00106
00107     return name;
00108 }
00109 }
```

**7.8.3.28 loadFocus()**

```
void Container::loadFocus (
    YAML::Node node ) [protected]
```

Definition at line 218 of file [constructor.cpp](#).

```
00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }
```

**7.8.3.29 loadName()**

```
bool Container::loadName (
    YAML::Node node ) [protected]
```

Definition at line 111 of file [constructor.cpp](#).

```
00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }
```

**7.8.3.30 loadSprites()**

```
void Container::loadSprites (
    YAML::Node node ) [protected]
```

Definition at line 122 of file [constructor.cpp](#).

```
00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])
00141         {
00142             float x, y, w, h;
00143             int repeat = 1;
00144             int gapX = 0;
00145             int gapY = 0;
00146
00147             int dx = 1;
00148             int dy = 1;
00149
00150             if(img["x"])
00151                 x = img["x"].as<float>() / 100.0;
00152             else x = 0;
00153             if(img["y"])
00154                 y = img["y"].as<float>() / 100.0;
00155             else y = 0;
00156             if(img["w"])
00157                 w = img["w"].as<float>() / 100.0;
00158             else w = 1;
00159             if(img["h"])
00160                 h = img["h"].as<float>() / 100.0;
00161             else h = 1;
```

```

00162         if(img["repeat"])
00163             repeat = img["repeat"].as<int>();
00164         if(img["gapX"])
00165             gapX = img["gapX"].as<int>();
00166         if(img["gapY"])
00167             gapY = img["gapY"].as<int>();
00168
00169         if(img["dx"])
00170             dx = img["dx"].as<int>();
00171         if(dx < 0) dx = -1;
00172         else dx = 1;
00173
00174         if(img["dy"])
00175             dy = img["dy"].as<int>();
00176         if(dy < 0) dy = -1;
00177         else dy = 1;
00178
00179         int imgw = image.width;
00180         int imgh = image.height;
00181
00182         if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183         {
00184             for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185             {
00186                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                 {
00188                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                     Image img2 = ImageFromImage(image, rect);
00190                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                     sprites.back().push_back(vis);
00193
00194                     UnloadImage(img2);
00195                 }
00196             }
00197         }else
00198         {
00199             for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200             {
00201                 for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202                 {
00203                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204                     Image img2 = ImageFromImage(image, rect);
00205                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207                     sprites.back().push_back(vis);
00208
00209                     UnloadImage(img2);
00210                 }
00211             }
00212         }
00213     }
00214     UnloadImage(image);
00215 }
00216 }

```

### 7.8.3.31 moveBy() [1/2]

```

void Frame::moveBy (
    fPoint rel ) [inherited]

```

Definition at line 65 of file [arithmetic.cpp](#).

```

00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }

```

### 7.8.3.32 moveBy() [2/2]

```

void Frame::moveBy (
    int x,
    int y ) [inherited]

```

Definition at line 75 of file [arithmetic.cpp](#).

```
00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }
```

### 7.8.3.33 moveCenterTo() [1/2]

```
void Frame::moveCenterTo (
    fPoint rel ) [inherited]
```

Definition at line 43 of file [arithmetic.cpp](#).

```
00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }
```

### 7.8.3.34 moveCenterTo() [2/2]

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

### 7.8.3.35 moveTo() [1/2]

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```



### 7.8.3.36 moveTo() [2/2]

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if (parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

### 7.8.3.37 nextImage()

```
void Container::nextImage ( )
```

move to next state of the sprite

Definition at line 247 of file [constructor.cpp](#).

```
00248 {
00249     if (sprites.empty()) return;
00250     focus[1]++;
00251     if (focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
```

### 7.8.3.38 nextSprite()

```
void Container::nextSprite ( )
```

move to the next sprite

Definition at line 261 of file [constructor.cpp](#).

```
00262 {
00263     if (sprites.empty()) return;
00264     focus[0]++;
00265     if (focus[0] >= sprites.size()) focus[0] = 0;
00266 }
```

### 7.8.3.39 operator fRect()

```
Frame::operator fRect ( ) const [inherited]
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

### 7.8.3.40 operator iRect()

```
Frame::operator iRect ( ) const [inherited]
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

## 7.8.3.41 operator Rectangle()

```
Frame::operator Rectangle ( ) const [inherited]
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

## 7.8.3.42 OverlappingArea() [1/2]

```
float Container::OverlappingArea (
    Container * container )
```

Definition at line 34 of file [overlap.cpp](#).

```
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if(w < 0 || h < 0) return 0;
00043     return w * h;
00044 }
```

## 7.8.3.43 OverlappingArea() [2/2]

```
float Container::OverlappingArea (
    Rectangle rec )
```

Definition at line 23 of file [overlap.cpp](#).

```
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
```

## 7.8.3.44 plug() [1/2]

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position

## Parameters

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
```

```

00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }

```

### 7.8.3.45 plug() [2/2]

```

void Frame::plug (
    Frame * par,
    fRect rel ) [inherited]

```

attach a frame to a parent by relative position

#### Parameters

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 12 of file [family.cpp](#).

```

00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }

```

### 7.8.3.46 prevImage()

```

void Container::prevImage ( )

```

move to previous state of the sprite

Definition at line 254 of file [constructor.cpp](#).

```

00255 {
00256     if(sprites.empty()) return;
00257     focus[1]--;
00258     if(focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }

```

### 7.8.3.47 prevSprite()

```

void Container::prevSprite ( )

```

move to the previous sprite

Definition at line 268 of file [constructor.cpp](#).

```

00269 {
00270     if(sprites.empty()) return;
00271     focus[0]--;
00272     if(focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }

```

**7.8.3.48 react()**

`Action * Container::react ( ) [virtual]`

Reimplemented in [ButtonImage](#), [Game](#), [Interface](#), and [Object](#).

Definition at line 36 of file [arithmetic.cpp](#).

```
00037 {
00038     return nullptr;
00039 }
```

**7.8.3.49 removeSubframe()**

`void Frame::removeSubframe (`  
     `Frame * subframe ) [protected], [inherited]`

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

**Parameters**

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

**7.8.3.50 resize() [1/2]**

`void Frame::resize (`  
     `fPoint rel ) [inherited]`

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

#### 7.8.3.51 `resize()` [2/2]

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if (parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

#### 7.8.3.52 `setProbability()`

```
void Container::setProbability (
    int prob )
```

Definition at line 280 of file [constructor.cpp](#).

```
00281 {
00282     probability = prob;
00283 }
```

#### 7.8.3.53 `setRelative()`

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

#### 7.8.3.54 `show()`

```
void Container::show ( )
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     visible = true;
00014 }
```

#### 7.8.3.55 `toggleVisibility()`

```
void Container::toggleVisibility ( )
```

Definition at line 21 of file [arithmetic.cpp](#).

```
00022 {
00023     visible = !visible;
00024 }
```

### 7.8.3.56 unplug()

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

### 7.8.3.57 updateFrame()

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual], [inherited]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
```

## 7.8.4 Friends And Related Symbol Documentation

### 7.8.4.1 changeImageAction

```
friend class changeImageAction [friend]
```

Definition at line 22 of file [container.hpp](#).

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

- [src/container/include/container.hpp](#)
- [src/container/src/arithmetic.cpp](#)
- [src/container/src/constructor.cpp](#)
- [src/container/src/destructor.cpp](#)
- [src/container/src/overlap.cpp](#)

## 7.9 Countdown Class Reference

count the time a playthrough takes

```
#include <countdown.hpp>
```

### Public Member Functions

- [CountDown](#) (int milliseconds)
- [~CountDown](#) ()
- int [get](#) ()
- bool [isFinished](#) ()
- void [run](#) ()

### 7.9.1 Detailed Description

count the time a playthrough takes

Definition at line 12 of file [countdown.hpp](#).

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 CountDown()

```
CountDown::CountDown (
    int milliseconds )
```

Definition at line 3 of file [countdown.cpp](#).

```
00004 {
00005     start = std::chrono::system_clock::now();
00006     finished = false;
00007     elapsed_seconds = std::chrono::milliseconds(milliseconds);
00008 }
```

### 7.9.2.2 ~CountDown()

```
CountDown::~CountDown ( )
```

Definition at line 10 of file [countdown.cpp](#).

```
00011 {
00012 }
```

## 7.9.3 Member Function Documentation

### 7.9.3.1 get()

```
int CountDown::get ( )
```

Definition at line 25 of file [countdown.cpp](#).

```
00026 {
00027     return elapsed_seconds.count() * 1000;
00028 }
```

### 7.9.3.2 isFinished()

```
bool Countdown::isFinished ( )
```

Definition at line 14 of file [countdown.cpp](#).

```
00015 {  
00016     return finished || (std::chrono::system_clock::now() - start) > elapsed_seconds;  
00017 }
```

### 7.9.3.3 run()

```
void Countdown::run ( )
```

Definition at line 19 of file [countdown.cpp](#).

```
00020 {  
00021     finished = false;  
00022     start = std::chrono::system_clock::now();  
00023 }
```

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

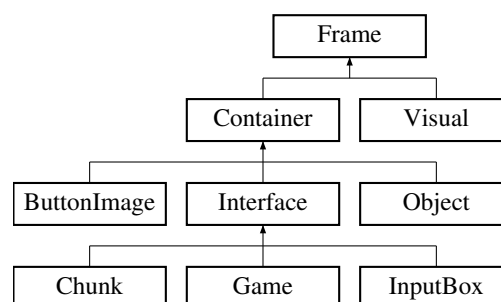
- [src/utls/include/countdown.hpp](#)
- [src/utls/src/countdown.cpp](#)

## 7.10 Frame Class Reference

position and size of object on screen

```
#include <frame.hpp>
```

Inheritance diagram for Frame:





## Public Member Functions

- [Frame](#) ([Frame](#) \*par, [Rectangle](#) rel)  
*create a frame with a parent and a relative position*
- [Frame](#) ([Frame](#) \*self)  
*clone a frame*
- [Frame](#) ([Rectangle](#) rec)  
*create a frame with a position and size*
- [~Frame](#) ()  
*destroy a frame*
- void [plug](#) ([Frame](#) \*par, [fRect](#) rel)  
*attach a frame to a parent by relative position*
- void [plug](#) ([Frame](#) \*par)  
*attach a frame to a parent by old relative position*
- void [unplug](#) ()  
*detach a frame from its parent*
- void [moveTo](#) ([fPoint](#) rel)
- void [moveTo](#) (int x, int y)
- void [moveCenterTo](#) ([fPoint](#) rel)
- void [moveCenterTo](#) (int x, int y)
- void [moveBy](#) ([fPoint](#) rel)
- void [moveBy](#) (int, int)
- void [resize](#) ([fPoint](#) rel)
- void [resize](#) (int w, int h)
- const [Rectangle](#) & [getFrame](#) () const
- const [fRect](#) & [getRelative](#) () const
- [Frame](#) \* [getParent](#) ()
- void [setRelative](#) ([fRect](#) rel)
- const [fPoint](#) & [getCenter](#) () const
- const float & [getX](#) () const
- const float & [getY](#) () const
- const float & [getW](#) () const
- const float & [getH](#) () const
- [operator Rectangle](#) () const
- [operator fRect](#) () const
- [operator iRect](#) () const

## Protected Member Functions

- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) ([Frame](#) \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) ([Frame](#) \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

### 7.10.1 Detailed Description

position and size of object on screen

when changing its position or size, it also changes position and size of all subframes

a subframe is relative to its parent by percentage (0.0f to 1.0f)

Definition at line 24 of file [frame.hpp](#).

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 Frame() [1/3]

```
Frame::Frame (
    Frame * par,
    Rectangle rel )
```

create a frame with a parent and a relative position

##### Parameters

<i>par</i>	parent <a href="#">Frame</a>
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 10 of file [constructor.cpp](#).

```
00011 {
00012     parent = nullptr;
00013     if(par == nullptr)
00014     {
00015         throw std::runtime_error("Frame::Frame(Frame* par, fRect rel): par is nullptr");
00016         return ;
00017     }
00018     parent = par;
00019     relative[0] = rel.x;
00020     relative[1] = rel.y;
00021     relative[2] = rel.width;
00022     relative[3] = rel.height;
00023
00024     parent->addSubframe(this);
00025
00026     updateFrame();
00027 }
```

#### 7.10.2.2 Frame() [2/3]

```
Frame::Frame (
    Frame * self )
```

clone a frame

##### Parameters

<i>self</i>	<a href="#">Frame</a> frame to clone
-------------	--------------------------------------

Definition at line 33 of file [constructor.cpp](#).

```

00034 {
00035     parent = nullptr;
00036     if(self == nullptr)
00037     {
00038         throw std::runtime_error("Frame::Frame(Frame* self): self is nullptr");
00039         return ;
00040     }
00041     parent = self->parent;
00042     relative = self->relative;
00043     frame = self->frame;
00044     for(auto& i : self->subframes)
00045     {
00046         subframes.push_back(i);
00047     }
00048 }

```

### 7.10.2.3 Frame() [3/3]

```

Frame::Frame (
    Rectangle rec )

```

create a frame with a position and size

This is a root frame

#### Parameters

<i>rec</i>	position and size in pixel
------------	----------------------------

Definition at line 57 of file [constructor.cpp](#).

```

00058 {
00059     parent = nullptr;
00060     frame = rec;
00061     parent = nullptr;
00062     relative = {1, 1, 1, 1};
00063 }

```

### 7.10.2.4 ~Frame()

```

Frame::~~Frame ( )

```

destroy a frame

MUST NOT DELETE ANYTHING

Definition at line 10 of file [destructor.cpp](#).

```

00011 {
00012 }

```

## 7.10.3 Member Function Documentation

### 7.10.3.1 addSubframe()

```

void Frame::addSubframe (
    Frame * subframe ) [protected]

```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

## Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```
00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }
```

### 7.10.3.2 beginUpdate()

```
void Frame::beginUpdate ( ) [protected]
```

Definition at line 113 of file [family.cpp](#).

```
00114 {
00115     mtx.lock();
00116 }
```

### 7.10.3.3 endUpdate()

```
void Frame::endUpdate ( ) [protected]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {
00120     mtx.unlock();
00121 }
```

### 7.10.3.4 getCenter()

```
const fPoint & Frame::getCenter ( ) const
```

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139
00140     return resu;
00141 }
```

### 7.10.3.5 getFrame()

```
const Rectangle & Frame::getFrame ( ) const
```

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {
00107     std::lock_guard<std::mutex> lock(mtx);
00108     return frame;
00109 }
```

### 7.10.3.6 getH()

```
const float & Frame::getH ( ) const
```

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```

### 7.10.3.7 getParent()

```
Frame * Frame::getParent ( )
```

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```

### 7.10.3.8 getRelative()

```
const fRect & Frame::getRelative ( ) const
```

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {  
00113     std::lock_guard<std::mutex> lock(mtx);  
00114     return relative;  
00115 }
```

### 7.10.3.9 getW()

```
const float & Frame::getW ( ) const
```

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {  
00157     std::lock_guard<std::mutex> lock(mtx);  
00158     return frame.width;  
00159 }
```

### 7.10.3.10 getX()

```
const float & Frame::getX ( ) const
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {  
00145     std::lock_guard<std::mutex> lock(mtx);  
00146     return frame.x;  
00147 }
```

### 7.10.3.11 getY()

```
const float & Frame::getY ( ) const
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {  
00151     std::lock_guard<std::mutex> lock(mtx);  
00152     return frame.y;  
00153 }
```

### 7.10.3.12 isroot()

```
bool Frame::isroot ( ) const [protected]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {  
00109     std::lock_guard<std::mutex> lock(mtx);  
00110     return parent == nullptr;  
00111 }
```

### 7.10.3.13 moveBy() [1/2]

```
void Frame::moveBy (  
    fPoint rel )
```

Definition at line 65 of file [arithmetic.cpp](#).

```
00066 {  
00067     if(isroot()) return ;  
00068     mtx.lock();  
00069     relative[0] += rel[0];  
00070     relative[1] += rel[1];  
00071     mtx.unlock();  
00072     updateFrame(true);  
00073 }
```

### 7.10.3.14 moveBy() [2/2]

```
void Frame::moveBy (  
    int x,  
    int y )
```

Definition at line 75 of file [arithmetic.cpp](#).

```
00076 {  
00077     if(parent != nullptr) return ;  
00078     mtx.lock();  
00079     frame.x += x;  
00080     frame.y += y;  
00081     mtx.unlock();  
00082     updateFrame(true);  
00083 }
```

### 7.10.3.15 moveCenterTo() [1/2]

```
void Frame::moveCenterTo (  
    fPoint rel )
```

Definition at line 43 of file [arithmetic.cpp](#).

```
00044 {  
00045     if(isroot()) return ;  
00046     mtx.lock();  
00047     fPoint center = getCenter();  
00048     relative[0] += rel[0] - center[0];  
00049     relative[1] += rel[1] - center[1];  
00050     mtx.unlock();  
00051     updateFrame(true);  
00052 }
```

**7.10.3.16 moveCenterTo() [2/2]**

```
void Frame::moveCenterTo (
    int x,
    int y )
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

**7.10.3.17 moveTo() [1/2]**

```
void Frame::moveTo (
    fPoint rel )
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

**7.10.3.18 moveTo() [2/2]**

```
void Frame::moveTo (
    int x,
    int y )
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.10.3.19 operator fRect()**

```
Frame::operator fRect ( ) const
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

**7.10.3.20 operator iRect()**

```
Frame::operator iRect ( ) const
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

**7.10.3.21 operator Rectangle()**

```
Frame::operator Rectangle ( ) const
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

**7.10.3.22 plug() [1/2]**

```
void Frame::plug (
    Frame * par )
```

attach a frame to a parent by old relative position

**Parameters**

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045     parent->addSubframe(this);
00046 }
00047 }
```

**7.10.3.23 plug() [2/2]**

```
void Frame::plug (
    Frame * par,
    fRect rel )
```

attach a frame to a parent by relative position

**Parameters**

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)



Definition at line 12 of file [family.cpp](#).

```
00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }
```

#### 7.10.3.24 removeSubframe()

```
void Frame::removeSubframe (
    Frame * subframe ) [protected]
```

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

##### Parameters

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

#### 7.10.3.25 resize() [1/2]

```
void Frame::resize (
    fPoint rel )
```

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

**7.10.3.26** `resize()` [2/2]

```
void Frame::resize (
    int w,
    int h )
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

**7.10.3.27** `setRelative()`

```
void Frame::setRelative (
    fRect rel )
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

**7.10.3.28** `unplug()`

```
void Frame::unplug ( )
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

**7.10.3.29** `updateFrame()`

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
```

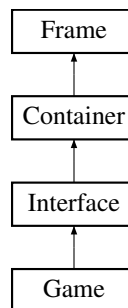
The documentation for this class was generated from the following files:

- [src/frame/include/frame.hpp](#)
- [src/frame/src/arithmetric.cpp](#)
- [src/frame/src/constructor.cpp](#)
- [src/frame/src/destructor.cpp](#)
- [src/frame/src/family.cpp](#)

## 7.11 Game Class Reference

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

Inheritance diagram for Game:



### Public Member Functions

- [Game](#) ([Frame](#) \*, [Rectangle](#))
- [Game](#) ([Game](#) \*)
- [Game](#) ([Game](#) \*, [Rectangle](#))
- [Game](#) ([Game](#) \*, [Frame](#) \*, [Rectangle](#))
- [~Game](#) ()
- [std::string linkContentAbsolute](#) ([std::string](#) path) override
- [Action \\* react](#) () override
- [Action \\* getRuntimeEvent](#) () override
- [void draw](#) () override
- [Container \\* getContainers](#) (int)
- [int getContainersSize](#) ()
- [std::string linkContent](#) ([std::string](#) path) override
- [std::string getName](#) ()
- [void setProbability](#) (int)
- [int getProbability](#) ()
- [void chooseSprite](#) (int)  
*choose a specific sprite from a vector of sprites*
- [void chooseImage](#) (int)  
*choose the state of the sprite*
- [void chooseImage](#) (int, int)  
*choose the state of the sprite*
- [void nextImage](#) ()  
*move to next state of the sprite*
- [void prevImage](#) ()  
*move to previous state of the sprite*
- [void nextSprite](#) ()

- *move to the next sprite*
- void `prevSprite` ()
- *move to the previous sprite*
- bool `isOverlapping` (fPoint)
- bool `isOverlapping` (Rectangle)
- bool `isOverlapping` (Container \*)
- float `OverlappingArea` (Rectangle)
- float `OverlappingArea` (Container \*)
- void `show` ()
- void `hide` ()
- void `toggleVisibility` ()
- bool `isVisible` ()
- int `getInstancelid` ()
- void `plug` (Frame \*par, fRect rel)
- *attach a frame to a parent by relative position*
- void `plug` (Frame \*par)
- *attach a frame to a parent by old relative position*
- void `unplug` ()
- *detach a frame from its parent*
- void `moveTo` (fPoint rel)
- void `moveTo` (int x, int y)
- void `moveCenterTo` (fPoint rel)
- void `moveCenterTo` (int x, int y)
- void `moveBy` (fPoint rel)
- void `moveBy` (int, int)
- void `resize` (fPoint rel)
- void `resize` (int w, int h)
- const Rectangle & `getFrame` () const
- const fRect & `getRelative` () const
- Frame \* `getParent` ()
- void `setRelative` (fRect rel)
- const fPoint & `getCenter` () const
- const float & `getX` () const
- const float & `getY` () const
- const float & `getW` () const
- const float & `getH` () const
- operator Rectangle () const
- operator fRect () const
- operator iRect () const

### Protected Member Functions

- void `loadChunk` (YAML::Node)
- void `loadCollide` (YAML::Node)
- void `loadEvent` (YAML::Node)
- void `loadAttactObject` (YAML::Node)
- void `loadMap` ()
- void `loadObject` (YAML::Node)
- void `loadControl` (YAML::Node)
- void `loadButton` (YAML::Node)
- void `drawNested` ()
- void `drawContainers` ()
- bool `loadName` (YAML::Node node)

- void [loadSprites](#) (YAML::Node node)
- void [loadFocus](#) (YAML::Node node)
- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) (Frame \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) (Frame \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

## Friends

- class [moveChunksAction](#)

## 7.11.1 Detailed Description

Definition at line 16 of file [game.hpp](#).

## 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 Game() [1/4]

```
Game::Game (
    Frame * frame,
    Rectangle rect )
```

Definition at line 9 of file [constructor.cpp](#).

```
00009                                     : Interface(frame, rect)
00010 {
00011     initState = true;
00012 }
```

### 7.11.2.2 Game() [2/4]

```
Game::Game (
    Game * other )
```

Definition at line 14 of file [constructor.cpp](#).

```
00014                                     : Interface(other)
00015 {
00016     initState = true;
00017 }
```

### 7.11.2.3 Game() [3/4]

```
Game::Game (
    Game * other,
    Rectangle rect )
```

Definition at line 19 of file [constructor.cpp](#).

```
00019                                     : Interface(other, rect)
00020 {
00021     initState = true;
00022 }
```

### 7.11.2.4 Game() [4/4]

```
Game::Game (
    Game * other,
    Frame * frame,
    Rectangle rect )
```

Definition at line 24 of file [constructor.cpp](#).

```
00024                                     : Interface(other, frame, rect)
00025 {
00026     initState = true;
00027 }
```

### 7.11.2.5 ~Game()

```
Game::~~Game ( )
```

Definition at line 4 of file [destructor.cpp](#).

```
00005 {
00006     for(auto &i : cache)
00007     {
00008         delete i;
00009     }
00010
00011     for(auto &i : chunks)
00012     {
00013         delete i;
00014     }
00015 }
```

## 7.11.3 Member Function Documentation

### 7.11.3.1 addSubframe()

```
void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]
```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```
00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }
```

### 7.11.3.2 beginUpdate()

```
void Frame::beginUpdate ( ) [protected], [inherited]
```

Definition at line 113 of file [family.cpp](#).

```
00114 {
00115     mtx.lock();
00116 }
```

**7.11.3.3 chooseImage() [1/2]**

```
void Container::chooseImage (
    int index ) [inherited]
```

choose the state of the sprite

Definition at line 231 of file [constructor.cpp](#).

```
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
```

**7.11.3.4 chooseImage() [2/2]**

```
void Container::chooseImage (
    int index,
    int index2 ) [inherited]
```

choose the state of the sprite

Definition at line 238 of file [constructor.cpp](#).

```
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
```

**7.11.3.5 chooseSprite()**

```
void Container::chooseSprite (
    int index ) [inherited]
```

choose a specific sprite from a vector of sprites

Definition at line 224 of file [constructor.cpp](#).

```
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
```

**7.11.3.6 draw()**

```
void Game::draw ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     drawNested();
00007
00008     for(auto i = chunks.begin(); i != chunks.end(); ++i)
00009     {
00010         (*i)->draw();
00011     }
00012
00013     drawContainers();
00014 }
```

**7.11.3.7 drawContainers()**

```
void Interface::drawContainers ( ) [protected], [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     for(auto& child : containers)
00014     {
00015         child->draw();
00016     }
00017 }
```

**7.11.3.8 drawNested()**

```
void Interface::drawNested ( ) [protected], [inherited]
```

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005     for(auto& child : nested)
00006     {
00007         child->draw();
00008     }
00009 }
```

**7.11.3.9 endUpdate()**

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {
00120     mtx.unlock();
00121 }
```

**7.11.3.10 getCenter()**

```
const fPoint & Frame::getCenter ( ) const [inherited]
```

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139
00140     return resu;
00141 }
```

**7.11.3.11 getContainers()**

```
Container * Interface::getContainers (
    int id ) [inherited]
```

Definition at line 176 of file [constructor.cpp](#).

```
00177 {
00178     if(id < 0 || id >= containers.size()) return nullptr;
00179     return containers[id];
00180 }
```



### 7.11.3.12 getContainersSize()

int Interface::getContainersSize ( ) [inherited]

Definition at line 182 of file [constructor.cpp](#).

```
00183 {  
00184     return containers.size();  
00185 }
```

### 7.11.3.13 getFrame()

const Rectangle & Frame::getFrame ( ) const [inherited]

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {  
00107     std::lock_guard<std::mutex> lock(mtx);  
00108     return frame;  
00109 }
```

### 7.11.3.14 getH()

const float & Frame::getH ( ) const [inherited]

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```

### 7.11.3.15 getInstanceId()

int Container::getInstanceId ( ) [inherited]

Definition at line 31 of file [arithmetic.cpp](#).

```
00032 {  
00033     return instance_id;  
00034 }
```

### 7.11.3.16 getName()

std::string Container::getName ( ) [inherited]

Definition at line 275 of file [constructor.cpp](#).

```
00276 {  
00277     return name;  
00278 }
```

### 7.11.3.17 getParent()

Frame \* Frame::getParent ( ) [inherited]

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```

**7.11.3.18 getProbability()**

```
int Container::getProbability ( ) [inherited]
```

Definition at line 285 of file [constructor.cpp](#).

```
00286 {
00287     return probability;
00288 }
```

**7.11.3.19 getRelative()**

```
const fRect & Frame::getRelative ( ) const [inherited]
```

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {
00113     std::lock_guard<std::mutex> lock(mtx);
00114     return relative;
00115 }
```

**7.11.3.20 getRuntimeEvent()**

```
Action * Game::getRuntimeEvent ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 8 of file [action.cpp](#).

```
00009 {
00010     // if now - mapSpeedClock < 10 millisecond, return nullptr
00011
00012     if(std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now() -
mapSpeedClock).count() < 20)
00013         return nullptr;
00014     Action* action;
00015     PacketAction* packet = nullptr;
00016     action = Interface::getRuntimeEvent();
00017
00018     if(action != nullptr)
00019     {
00020         packet = new PacketAction();
00021         packet->addAction(action);
00022     }
00023
00024     action = new moveChunksAction(this, mapDisplacement);
00025     if(packet == nullptr) packet = new PacketAction();
00026     packet->addAction(action);
00027
00028     for(auto i : chunks)
00029     {
00030         Action* act = i->getRuntimeEvent();
00031         if(act == nullptr)
00032             continue;
00033         if(packet == nullptr)
00034             packet = new PacketAction();
00035         packet->addAction(act);
00036     }
00037
00038     action = new moveObjectAction(main, mapDisplacement);
00039     if(packet == nullptr) packet = new PacketAction();
00040     packet->addAction(action);
00041
00042     mapSpeedClock = std::chrono::system_clock::now();
00043
00044     return packet;
00045 }
```

### 7.11.3.21 getW()

```
const float & Frame::getW ( ) const [inherited]
```

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {  
00157     std::lock_guard<std::mutex> lock(mtx);  
00158     return frame.width;  
00159 }
```

### 7.11.3.22 getX()

```
const float & Frame::getX ( ) const [inherited]
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {  
00145     std::lock_guard<std::mutex> lock(mtx);  
00146     return frame.x;  
00147 }
```

### 7.11.3.23 getY()

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {  
00151     std::lock_guard<std::mutex> lock(mtx);  
00152     return frame.y;  
00153 }
```

### 7.11.3.24 hide()

```
void Container::hide ( ) [inherited]
```

Definition at line 16 of file [arithmetic.cpp](#).

```
00017 {  
00018     visible = false;  
00019 }
```

### 7.11.3.25 isOverlapping() [1/3]

```
bool Container::isOverlapping (  
    Container * container ) [inherited]
```

Definition at line 16 of file [overlap.cpp](#).

```
00017 {  
00018     Rectangle rec = getFrame();  
00019     Rectangle rec2 = container->getFrame();  
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +  
        rec2.height && rec.y + rec.height >= rec2.y);  
00021 }
```

**7.11.3.26 isOverlapping() [2/3]**

```
bool Container::isOverlapping (
    fPoint point ) [inherited]
```

Definition at line 3 of file [overlap.cpp](#).

```
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
        rec.y + rec.height);
00007
00008 }
```

**7.11.3.27 isOverlapping() [3/3]**

```
bool Container::isOverlapping (
    Rectangle rec ) [inherited]
```

Definition at line 10 of file [overlap.cpp](#).

```
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
        rec2.height && rec.y + rec.height >= rec2.y);
00014 }
```

**7.11.3.28 isroot()**

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

**7.11.3.29 isVisible()**

```
bool Container::isVisible ( ) [inherited]
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     return visible;
00029 }
```

**7.11.3.30 linkContent()**

```
std::string Interface::linkContent (
    std::string path ) [override], [virtual], [inherited]
```

Reimplemented from [Container](#).

Definition at line 78 of file [constructor.cpp](#).

```
00079 {
00080     return linkContentAbsolute(PATB::INTERFACE_ + path);
00081 }
```

### 7.11.3.31 linkContentAbsolute()

```
std::string Game::linkContentAbsolute (
    std::string path ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 29 of file [constructor.cpp](#).

```
00030 {
00031     YAML::Node node = YAML_FILE::readFile(path);
00032     if(!loadName(node)) return "";
00033
00034     if(node["textures"])
00035         loadSprites(node["textures"]);
00036
00037     if(node["focus"])
00038         loadFocus(node["focus"]);
00039     else chooseImage(0, 0);
00040
00041     if(node["object"])
00042     {
00043         loadObject(node["object"]);
00044         for(int i = 0; i < getContainersSize(); i++)
00045             getContainers(i)->hide();
00046         main = getContainers(0);
00047         main->show();
00048     }
00049     if(node["collide"])
00050         loadCollide(node["collide"]);
00051
00052     if(node["chunk"])
00053         loadChunk(node["chunk"]);
00054
00055     if(node["attach-object"])
00056         loadAttactObject(node["attach-object"]);
00057
00058     if(node["control"])
00059         loadControl(node["control"]);
00060
00061     if(node["event"])
00062         loadEvent(node["event"]);
00063
00064     if(node["button"])
00065         loadButton(node["button"]);
00066     return getName();
00067 }
```

### 7.11.3.32 loadAttactObject()

```
void Game::loadAttactObject (
    YAML::Node node ) [protected]
```

Definition at line 143 of file [constructor.cpp](#).

```
00144 {
00145     for(auto i : node)
00146     {
00147         int id = i["chunk"].as<int>();
00148         int objID = i["object"][0].as<int>();
00149         int prob = i["object"][1].as<int>();
00150         Container* container = getContainers(objID);
00151         container->setProbability(prob);
00152         cache[id]->addVisitor(container);
00153     }
00154 }
```

### 7.11.3.33 loadButton()

```
void Interface::loadButton (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 159 of file [constructor.cpp](#).

```

00160 {
00161     for(auto i : node)
00162     {
00163         Rectangle rel({0, 0, 0, 0});
00164         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00165         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00166         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00167         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00168         ButtonImage *obj;
00169         obj = new ButtonImage(this, rel);
00170         obj->linkContent(i["path"].as<std::string>());
00171         obj->show();
00172         containers.push_back(obj);
00173     }
00174 }

```

### 7.11.3.34 loadChunk()

```

void Game::loadChunk (
    YAML::Node node ) [protected]

```

Definition at line 113 of file [constructor.cpp](#).

```

00114 {
00115     for(auto i : node)
00116     {
00117         float x = 0, y = 0, w = 1, h = 1;
00118         int repeat = 1;
00119         std::string path = i["file"].as<std::string>();
00120         if(i["x"]) x = i["x"].as<float>() / 100;
00121         if(i["y"]) y = i["y"].as<float>() / 100;
00122         if(i["w"]) w = i["w"].as<float>() / 100;
00123         if(i["h"]) h = i["h"].as<float>() / 100;
00124         if(i["repeat"]) repeat = i["repeat"].as<int>();
00125         fPoint direction = {1, 0};
00126         float velo = 0.002;
00127         if(i["velocity"])
00128         {
00129             velo = i["velocity"][0].as<float>();
00130             direction = {i["velocity"][1].as<float>(), i["velocity"][2].as<float>()};
00131         }
00132         float angle = VECTOR2D::getAngle(direction);
00133         fPoint displacement = {velo * cos(angle), velo * sin(angle)};
00134         Chunk* chunk = new Chunk(this, {x, y, w, h});
00135         chunk->linkContent(path);
00136         chunk->setVelocity(displacement);
00137         cache.push_back(chunk);
00138         while(--repeat > 0)
00139             cache.push_back(new Chunk(cache[0]));
00140     }
00141 }

```

### 7.11.3.35 loadCollide()

```

void Game::loadCollide (
    YAML::Node node ) [protected]

```

Definition at line 69 of file [constructor.cpp](#).

```

00070 {
00071 }

```

### 7.11.3.36 loadControl()

```

void Interface::loadControl (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 134 of file [constructor.cpp](#).

```

00135 {
00136     for(auto stroke : node)
00137     {

```

```

00138         KeyStroke* k = new KeyStroke();
00139         for(auto key : stroke["key"])
00140         {
00141             k->add(toKey(key.as<std::string>()));
00142         }
00143         std::string action = stroke["action"].as<std::string>();
00144
00145         if(action == "move-object")
00146         {
00147             int id = stroke["args"][0].as<int>();
00148             float v = stroke["args"][1].as<float>() / 100.0;
00149             float x = stroke["args"][2].as<float>();
00150             float y = stroke["args"][3].as<float>();
00151             moveObjectAction* action = new moveObjectAction(containers[id], fPoint({x, y}), v);
00152             k->addAction(action);
00153         }
00154
00155         keystrokes.push_back(k);
00156     }
00157 }

```

### 7.11.3.37 loadEvent()

```

void Game::loadEvent (
    YAML::Node node ) [protected]

```

Definition at line 156 of file [constructor.cpp](#).

```

00157 {
00158     if(node["map-speed"])
00159     {
00160         mapSpeed = node["map-speed"].as<float>();
00161     }
00162     if(node["map-direction"])
00163     {
00164         mapDirection[0] = node["map-direction"][0].as<float>();
00165         mapDirection[1] = node["map-direction"][1].as<float>();
00166     }
00167     float angle = VECTOR2D::getAngle(mapDirection);
00168     std::cout << "hehe: " << angle << std::endl;
00169     mapDisplacement[0] = mapSpeed * cos(angle);
00170     mapDisplacement[1] = mapSpeed * sin(angle);
00171 }

```

### 7.11.3.38 loadFocus()

```

void Container::loadFocus (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 218 of file [constructor.cpp](#).

```

00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }

```

### 7.11.3.39 loadMap()

```

void Game::loadMap ( ) [protected]

```

Definition at line 73 of file [constructor.cpp](#).

```

00074 {
00075     if(cache.empty()) return ;
00076     while(!chunks.empty())
00077     {
00078         fRect rec = chunks.back()->getRelative();
00079         if(rec[1] > 1) chunks.pop_back();
00080         else break;
00081     }
00082     if(chunks.empty())
00083     {

```

```

00084     Rectangle rel;
00085     rel.width = cache[0]->getRelative()[2];
00086     rel.height = cache[0]->getRelative()[3];
00087     rel.x = 0;
00088     rel.y = (1.01 - rel.height);
00089
00090     Chunk* chunk = new Chunk(cache[0], this, rel);
00091     chunks.push_front(chunk);
00092     for(int i = 0; i < 3; i++)
00093     {
00094         rel.y += 0.005 - rel.height;
00095         chunk = new Chunk(cache[0], this, rel);
00096         chunks.push_front(chunk);
00097     }
00098 }
00099 while(chunks.front()->getRelative()[1] > 0)
00100 {
00101     Rectangle rel;
00102     rel.width = chunks.front()->getRelative()[2];
00103     rel.height = chunks.front()->getRelative()[3];
00104     rel.x = 0;
00105     rel.y = (chunks.front()->getRelative()[1] + 0.005 - rel.height);
00106
00107     int id = GetRandomValue(0, cache.size() - 1);
00108     Chunk* chunk = new Chunk(cache[id], this, rel);
00109     chunks.push_front(chunk);
00110 }
00111 }

```

#### 7.11.3.40 loadName()

```

bool Container::loadName (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 111 of file [constructor.cpp](#).

```

00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }

```

#### 7.11.3.41 loadObject()

```

void Interface::loadObject (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 117 of file [constructor.cpp](#).

```

00118 {
00119     for(auto i : node)
00120     {
00121         Rectangle rel({0, 0, 0, 0});
00122         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00123         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00124         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00125         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00126         Container *obj;
00127         obj = new Object(this, rel);
00128         obj->linkContent(i["path"].as<std::string>());
00129         containers.push_back(obj);
00130     }
00131 }

```



### 7.11.3.42 loadSprites()

```
void Container::loadSprites (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 122 of file [constructor.cpp](#).

```
00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])
00141         {
00142             float x, y, w, h;
00143             int repeat = 1;
00144             int gapX = 0;
00145             int gapY = 0;
00146
00147             int dx = 1;
00148             int dy = 1;
00149
00150             if(img["x"])
00151                 x = img["x"].as<float>() / 100.0;
00152             else x = 0;
00153             if(img["y"])
00154                 y = img["y"].as<float>() / 100.0;
00155             else y = 0;
00156             if(img["w"])
00157                 w = img["w"].as<float>() / 100.0;
00158             else w = 1;
00159             if(img["h"])
00160                 h = img["h"].as<float>() / 100.0;
00161             else h = 1;
00162             if(img["repeat"])
00163                 repeat = img["repeat"].as<int>();
00164             if(img["gapX"])
00165                 gapX = img["gapX"].as<int>();
00166             if(img["gapY"])
00167                 gapY = img["gapY"].as<int>();
00168
00169             if(img["dx"])
00170                 dx = img["dx"].as<int>();
00171             if(dx < 0) dx = -1;
00172             else dx = 1;
00173
00174             if(img["dy"])
00175                 dy = img["dy"].as<int>();
00176             if(dy < 0) dy = -1;
00177             else dy = 1;
00178
00179             int imgw = image.width;
00180             int imgh = image.height;
00181
00182             if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183             {
00184                 for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185                 {
00186                     for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                     {
00188                         Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                         Image img2 = ImageFromImage(image, rect);
00190                         Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                         Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                         sprites.back().push_back(vis);
00193
00194                         UnloadImage(img2);
00195                     }
00196                 }
00197             }else
00198             {
00199                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200                 {
```

```

00201         for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202         {
00203             Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204             Image img2 = ImageFromImage(image, rect);
00205             Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206             Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207             sprites.back().push_back(vis);
00208
00209             UnloadImage(img2);
00210         }
00211     }
00212 }
00213 }
00214     UnloadImage(image);
00215 }
00216 }

```

#### 7.11.3.43 moveBy() [1/2]

```

void Frame::moveBy (
    fPoint rel ) [inherited]

```

Definition at line 65 of file [arithmetic.cpp](#).

```

00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }

```

#### 7.11.3.44 moveBy() [2/2]

```

void Frame::moveBy (
    int x,
    int y ) [inherited]

```

Definition at line 75 of file [arithmetic.cpp](#).

```

00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }

```

#### 7.11.3.45 moveCenterTo() [1/2]

```

void Frame::moveCenterTo (
    fPoint rel ) [inherited]

```

Definition at line 43 of file [arithmetic.cpp](#).

```

00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }

```

**7.11.3.46 moveCenterTo()** [2/2]

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

**7.11.3.47 moveTo()** [1/2]

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

**7.11.3.48 moveTo()** [2/2]

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.11.3.49 nextImage()**

```
void Container::nextImage ( ) [inherited]
```

move to next state of the sprite

Definition at line 247 of file [constructor.cpp](#).

```
00248 {
00249     if(sprites.empty()) return;
00250     focus[1]++;
00251     if(focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
```

**7.11.3.50 nextSprite()**

```
void Container::nextSprite ( ) [inherited]
```

move to the next sprite

Definition at line 261 of file [constructor.cpp](#).

```
00262 {
00263     if (sprites.empty()) return;
00264     focus[0]++;
00265     if (focus[0] >= sprites.size()) focus[0] = 0;
00266 }
```

**7.11.3.51 operator fRect()**

```
Frame::operator fRect ( ) const [inherited]
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

**7.11.3.52 operator iRect()**

```
Frame::operator iRect ( ) const [inherited]
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

**7.11.3.53 operator Rectangle()**

```
Frame::operator Rectangle ( ) const [inherited]
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

**7.11.3.54 OverlappingArea() [1/2]**

```
float Container::OverlappingArea (
    Container * container ) [inherited]
```

Definition at line 34 of file [overlap.cpp](#).

```
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if (w < 0 || h < 0) return 0;
00043     return w * h;
00044 }
```

### 7.11.3.55 OverlappingArea() [2/2]

```
float Container::OverlappingArea (
    Rectangle rec ) [inherited]
```

Definition at line 23 of file [overlap.cpp](#).

```
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
```

### 7.11.3.56 plug() [1/2]

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position

#### Parameters

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
```

### 7.11.3.57 plug() [2/2]

```
void Frame::plug (
    Frame * par,
    fRect rel ) [inherited]
```

attach a frame to a parent by relative position

#### Parameters

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 12 of file [family.cpp](#).

```
00013 {
00014     if(par == nullptr)
00015     {
```

```

00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }

```

### 7.11.3.58 prevImage()

void Container::prevImage ( ) [inherited]

move to previous state of the sprite

Definition at line 254 of file [constructor.cpp](#).

```

00255 {
00256     if (sprites.empty()) return;
00257     focus[1]--;
00258     if (focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }

```

### 7.11.3.59 prevSprite()

void Container::prevSprite ( ) [inherited]

move to the previous sprite

Definition at line 268 of file [constructor.cpp](#).

```

00269 {
00270     if (sprites.empty()) return;
00271     focus[0]--;
00272     if (focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }

```

### 7.11.3.60 react()

Action \* Game::react ( ) [override], [virtual]

Reimplemented from [Container](#).

Definition at line 3 of file [action.cpp](#).

```

00004 {
00005     return Interface::react();
00006 }

```

### 7.11.3.61 removeSubframe()

void Frame::removeSubframe (   
     Frame \* subframe ) [protected], [inherited]

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

## Parameters

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

### 7.11.3.62 `resize()` [1/2]

```
void Frame::resize (
    fPoint rel ) [inherited]
```

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

### 7.11.3.63 `resize()` [2/2]

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

### 7.11.3.64 `setProbability()`

```
void Container::setProbability (
    int prob ) [inherited]
```

Definition at line 280 of file [constructor.cpp](#).

```
00281 {
00282     probability = prob;
00283 }
```

**7.11.3.65 setRelative()**

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

**7.11.3.66 show()**

```
void Container::show ( ) [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     visible = true;
00014 }
```

**7.11.3.67 toggleVisibility()**

```
void Container::toggleVisibility ( ) [inherited]
```

Definition at line 21 of file [arithmetic.cpp](#).

```
00022 {
00023     visible = !visible;
00024 }
```

**7.11.3.68 unplug()**

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

**7.11.3.69 updateFrame()**

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual], [inherited]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
```



## 7.11.4 Friends And Related Symbol Documentation

### 7.11.4.1 moveChunksAction

```
friend class moveChunksAction [friend]
```

Definition at line 19 of file [game.hpp](#).

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

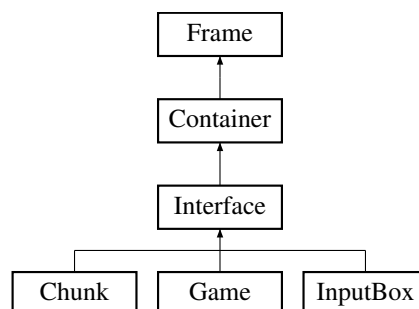
- [src/game/include/game.hpp](#)
- [src/game/src/action.cpp](#)
- [src/game/src/arithmetic.cpp](#)
- [src/game/src/constructor.cpp](#)
- [src/game/src/destructor.cpp](#)

## 7.12 Interface Class Reference

where user can interact with the game

```
#include <interface.hpp>
```

Inheritance diagram for Interface:



### Public Member Functions

- [Interface](#) ([Frame](#) \*, [Rectangle](#))
- [Interface](#) ([Interface](#) \*)
- [Interface](#) ([Interface](#) \*, [Rectangle](#))
- [Interface](#) ([Interface](#) \*, [Frame](#) \*, [Rectangle](#))
- [~Interface](#) ()
- [Container](#) \* [getContainers](#) (int)
- int [getContainersSize](#) ()
- std::string [linkContent](#) (std::string path) override
- std::string [linkContentAbsolute](#) (std::string path) override
- [Action](#) \* [react](#) () override
- [Action](#) \* [getRuntimeEvent](#) () override
- void [draw](#) () override
- std::string [getName](#) ()
- void [setProbability](#) (int)

- int [getProbability](#) ()
- void [chooseSprite](#) (int)
  - choose a specific sprite from a vector of sprites*
- void [chooseImage](#) (int)
  - choose the state of the sprite*
- void [chooseImage](#) (int, int)
  - choose the state of the sprite*
- void [nextImage](#) ()
  - move to next state of the sprite*
- void [prevImage](#) ()
  - move to previous state of the sprite*
- void [nextSprite](#) ()
  - move to the next sprite*
- void [prevSprite](#) ()
  - move to the previous sprite*
- bool [isOverlapping](#) (fPoint)
- bool [isOverlapping](#) (Rectangle)
- bool [isOverlapping](#) (Container \*)
- float [OverlappingArea](#) (Rectangle)
- float [OverlappingArea](#) (Container \*)
- void [show](#) ()
- void [hide](#) ()
- void [toggleVisibility](#) ()
- bool [isVisible](#) ()
- int [getInstanceld](#) ()
- void [plug](#) (Frame \*par, fRect rel)
  - attach a frame to a parent by relative position*
- void [plug](#) (Frame \*par)
  - attach a frame to a parent by old relative position*
- void [unplug](#) ()
  - detach a frame from its parent*
- void [moveTo](#) (fPoint rel)
- void [moveTo](#) (int x, int y)
- void [moveCenterTo](#) (fPoint rel)
- void [moveCenterTo](#) (int x, int y)
- void [moveBy](#) (fPoint rel)
- void [moveBy](#) (int, int)
- void [resize](#) (fPoint rel)
- void [resize](#) (int w, int h)
- const Rectangle & [getFrame](#) () const
- const fRect & [getRelative](#) () const
- Frame \* [getParent](#) ()
- void [setRelative](#) (fRect rel)
- const fPoint & [getCenter](#) () const
- const float & [getX](#) () const
- const float & [getY](#) () const
- const float & [getW](#) () const
- const float & [getH](#) () const
- operator Rectangle () const
- operator fRect () const
- operator iRect () const

## Protected Member Functions

- void [loadObject](#) (YAML::Node)
- void [loadControl](#) (YAML::Node)
- void [loadButton](#) (YAML::Node)
- void [drawNested](#) ()
- void [drawContainers](#) ()
- bool [loadName](#) (YAML::Node node)
- void [loadSprites](#) (YAML::Node node)
- void [loadFocus](#) (YAML::Node node)
- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) ([Frame](#) \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) ([Frame](#) \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

## Friends

- class [moveObjectAction](#)

## 7.12.1 Detailed Description

where user can interact with the game

manages containers, all actions, subframes etc.

Definition at line 20 of file [interface.hpp](#).

## 7.12.2 Constructor & Destructor Documentation

### 7.12.2.1 Interface() [1/4]

```
Interface::Interface (
    Frame * frame,
    Rectangle rect )
```

Definition at line 8 of file [constructor.cpp](#).

```
00008                                     : Container(frame, rect)
00009 {
00010 }
```

## 7.12.2.2 Interface() [2/4]

```
Interface::Interface (
    Interface * other )
```

Definition at line 12 of file [constructor.cpp](#).

```
00012                                     : Container(other)
00013 {
00014     for(auto i : other->nested)
00015     {
00016         Rectangle rel;
00017         rel.x = i->getRelative()[0];
00018         rel.y = i->getRelative()[1];
00019         rel.width = i->getRelative()[2];
00020         rel.height = i->getRelative()[3];
00021         nested.push_back(new Interface(i, this, rel));
00022     }
00023     for(auto i : other->containers)
00024     {
00025         Rectangle rel;
00026         rel.x = i->getRelative()[0];
00027         rel.y = i->getRelative()[1];
00028         rel.width = i->getRelative()[2];
00029         rel.height = i->getRelative()[3];
00030         containers.push_back(new Container(i, this, rel));
00031     }
00032 }
```

## 7.12.2.3 Interface() [3/4]

```
Interface::Interface (
    Interface * other,
    Rectangle rect )
```

Definition at line 34 of file [constructor.cpp](#).

```
00034                                     : Container(other, rect)
00035 {
00036     for(auto i : other->nested)
00037     {
00038         Rectangle rel;
00039         rel.x = i->getRelative()[0];
00040         rel.y = i->getRelative()[1];
00041         rel.width = i->getRelative()[2];
00042         rel.height = i->getRelative()[3];
00043         nested.push_back(new Interface(i, this, rel));
00044     }
00045     for(auto i : other->containers)
00046     {
00047         Rectangle rel;
00048         rel.x = i->getRelative()[0];
00049         rel.y = i->getRelative()[1];
00050         rel.width = i->getRelative()[2];
00051         rel.height = i->getRelative()[3];
00052         containers.push_back(new Container(i, this, rel));
00053     }
00054 }
```

## 7.12.2.4 Interface() [4/4]

```
Interface::Interface (
    Interface * other,
    Frame * frame,
    Rectangle rect )
```

Definition at line 56 of file [constructor.cpp](#).

```
00056                                     : Container(other, frame, rect)
00057 {
00058     for(auto i : other->nested)
00059     {
00060         Rectangle rel;
```

```

00061         rel.x = i->getRelative()[0];
00062         rel.y = i->getRelative()[1];
00063         rel.width = i->getRelative()[2];
00064         rel.height = i->getRelative()[3];
00065         nested.push_back(new Interface(i, this, rel));
00066     }
00067     for(auto i : other->containers)
00068     {
00069         Rectangle rel;
00070         rel.x = i->getRelative()[0];
00071         rel.y = i->getRelative()[1];
00072         rel.width = i->getRelative()[2];
00073         rel.height = i->getRelative()[3];
00074         containers.push_back(new Container(i, this, rel));
00075     }
00076 }

```

### 7.12.2.5 ~Interface()

Interface::~Interface ( )

Definition at line 4 of file [destructor.cpp](#).

```

00005 {
00006     for (auto& i : containers)
00007         delete i;
00008     containers.clear();
00009
00010     for (auto& i : nested)
00011         delete i;
00012     nested.clear();
00013
00014     for (auto& i : keystrokes)
00015         delete i;
00016 }

```

## 7.12.3 Member Function Documentation

### 7.12.3.1 addSubframe()

```

void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]

```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```

00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }

```

### 7.12.3.2 beginUpdate()

```

void Frame::beginUpdate ( ) [protected], [inherited]

```

Definition at line 113 of file [family.cpp](#).

```

00114 {
00115     mtx.lock();
00116 }

```

**7.12.3.3 chooseImage() [1/2]**

```
void Container::chooseImage (
    int index ) [inherited]
```

choose the state of the sprite

Definition at line 231 of file [constructor.cpp](#).

```
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
```

**7.12.3.4 chooseImage() [2/2]**

```
void Container::chooseImage (
    int index,
    int index2 ) [inherited]
```

choose the state of the sprite

Definition at line 238 of file [constructor.cpp](#).

```
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
```

**7.12.3.5 chooseSprite()**

```
void Container::chooseSprite (
    int index ) [inherited]
```

choose a specific sprite from a vector of sprites

Definition at line 224 of file [constructor.cpp](#).

```
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
```

**7.12.3.6 draw()**

```
void Interface::draw ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 19 of file [arithmetic.cpp](#).

```
00020 {
00021     Container::draw();
00022
00023     drawNested();
00024
00025     drawContainers();
00026
00027
00028 }
```

### 7.12.3.7 drawContainers()

```
void Interface::drawContainers ( ) [protected]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     for(auto& child : containers)
00014     {
00015         child->draw();
00016     }
00017 }
```

### 7.12.3.8 drawNested()

```
void Interface::drawNested ( ) [protected]
```

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005     for(auto& child : nested)
00006     {
00007         child->draw();
00008     }
00009 }
```

### 7.12.3.9 endUpdate()

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {
00120     mtx.unlock();
00121 }
```

### 7.12.3.10 getCenter()

```
const fPoint & Frame::getCenter ( ) const [inherited]
```

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139
00140     return resu;
00141 }
```

### 7.12.3.11 getContainers()

```
Container * Interface::getContainers (
    int id )
```

Definition at line 176 of file [constructor.cpp](#).

```
00177 {
00178     if(id < 0 || id >= containers.size()) return nullptr;
00179     return containers[id];
00180 }
```

### 7.12.3.12 getContainersSize()

```
int Interface::getContainersSize ( )
```

Definition at line 182 of file [constructor.cpp](#).

```
00183 {  
00184     return containers.size();  
00185 }
```

### 7.12.3.13 getFrame()

```
const Rectangle & Frame::getFrame ( ) const [inherited]
```

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {  
00107     std::lock_guard<std::mutex> lock(mtx);  
00108     return frame;  
00109 }
```

### 7.12.3.14 getH()

```
const float & Frame::getH ( ) const [inherited]
```

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```

### 7.12.3.15 getInstanceId()

```
int Container::getInstanceId ( ) [inherited]
```

Definition at line 31 of file [arithmetic.cpp](#).

```
00032 {  
00033     return instance_id;  
00034 }
```

### 7.12.3.16 getName()

```
std::string Container::getName ( ) [inherited]
```

Definition at line 275 of file [constructor.cpp](#).

```
00276 {  
00277     return name;  
00278 }
```

### 7.12.3.17 getParent()

```
Frame * Frame::getParent ( ) [inherited]
```

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```



### 7.12.3.18 getProbability()

int Container::getProbability ( ) [inherited]

Definition at line 285 of file [constructor.cpp](#).

```
00286 {
00287     return probability;
00288 }
```

### 7.12.3.19 getRelative()

const fRect & Frame::getRelative ( ) const [inherited]

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {
00113     std::lock_guard<std::mutex> lock(mtx);
00114     return relative;
00115 }
```

### 7.12.3.20 getRuntimeEvent()

Action \* Interface::getRuntimeEvent ( ) [override], [virtual]

Reimplemented from [Container](#).

Definition at line 4 of file [action.cpp](#).

```
00005 {
00006     PacketAction* packet = nullptr;
00007     Action* action = Container::getRuntimeEvent();
00008
00009     if(action != nullptr)
00010     {
00011         packet = new PacketAction();
00012         packet->addAction(action);
00013     }
00014
00015     for(auto i : nested)
00016     {
00017         action = i->getRuntimeEvent();
00018         if(action != nullptr)
00019         {
00020             if(packet == nullptr) packet = new PacketAction();
00021             packet->addAction(action);
00022         }
00023     }
00024
00025     for(auto i : containers)
00026     {
00027         action = i->getRuntimeEvent();
00028         if(action != nullptr)
00029         {
00030             if(packet == nullptr) packet = new PacketAction();
00031             packet->addAction(action);
00032         }
00033     }
00034
00035     return packet;
00036 }
```

### 7.12.3.21 getW()

const float & Frame::getW ( ) const [inherited]

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {
00157     std::lock_guard<std::mutex> lock(mtx);
00158     return frame.width;
00159 }
```

**7.12.3.22 getX()**

```
const float & Frame::getX ( ) const [inherited]
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {
00145     std::lock_guard<std::mutex> lock(mtx);
00146     return frame.x;
00147 }
```

**7.12.3.23 getY()**

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {
00151     std::lock_guard<std::mutex> lock(mtx);
00152     return frame.y;
00153 }
```

**7.12.3.24 hide()**

```
void Container::hide ( ) [inherited]
```

Definition at line 16 of file [arithmetic.cpp](#).

```
00017 {
00018     visible = false;
00019 }
```

**7.12.3.25 isOverlapping() [1/3]**

```
bool Container::isOverlapping (
    Container * container ) [inherited]
```

Definition at line 16 of file [overlap.cpp](#).

```
00017 {
00018     Rectangle rec = getFrame();
00019     Rectangle rec2 = container->getFrame();
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
    rec2.height && rec.y + rec.height >= rec2.y);
00021 }
```

**7.12.3.26 isOverlapping() [2/3]**

```
bool Container::isOverlapping (
    fPoint point ) [inherited]
```

Definition at line 3 of file [overlap.cpp](#).

```
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
    rec.y + rec.height);
00007
00008 }
```

### 7.12.3.27 isOverlapping() [3/3]

```
bool Container::isOverlapping (
    Rectangle rec ) [inherited]
```

Definition at line 10 of file [overlap.cpp](#).

```
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
00014             rec2.height && rec.y + rec.height >= rec2.y);
00014 }
```

### 7.12.3.28 isroot()

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

### 7.12.3.29 isVisible()

```
bool Container::isVisible ( ) [inherited]
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     return visible;
00029 }
```

### 7.12.3.30 linkContent()

```
std::string Interface::linkContent (
    std::string path ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 78 of file [constructor.cpp](#).

```
00079 {
00080     return linkContentAbsolute(PATB::INTERFACE_ + path);
00081 }
```

**7.12.3.31 linkContentAbsolute()**

```
std::string Interface::linkContentAbsolute (
    std::string path ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 83 of file [constructor.cpp](#).

```
00084 {
00085     YAML::Node node = YAML_FILE::readFile(path);
00086     if(!loadName(node)) return "";
00087
00088     if(node["textures"])
00089         loadSprites(node["textures"]);
00090
00091     if(node["focus"])
00092         loadFocus(node["focus"]);
00093     else chooseImage(0, 0);
00094
00095     if(node["object"])
00096         loadObject(node["object"]);
00097
00098     if(node["control"])
00099         loadControl(node["control"]);
00100
00101     if(node["button"])
00102         loadButton(node["button"]);
00103
00104     // if(node["collide"])
00105     //     loadCollide(node["collide"]);
00106
00107     // if(node["chunk"])
00108     //     loadChunk(node["chunk"]);
00109
00110
00111     // if(node["event"])
00112     //     loadEvent(node["event"]);
00113
00114     return getName();
00115 }
```

**7.12.3.32 loadButton()**

```
void Interface::loadButton (
    YAML::Node node ) [protected]
```

Definition at line 159 of file [constructor.cpp](#).

```
00160 {
00161     for(auto i : node)
00162     {
00163         Rectangle rel({0, 0, 0, 0});
00164         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00165         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00166         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00167         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00168         ButtonImage *obj;
00169         obj = new ButtonImage(this, rel);
00170         obj->linkContent(i["path"].as<std::string>());
00171         obj->show();
00172         containers.push_back(obj);
00173     }
00174 }
```

**7.12.3.33 loadControl()**

```
void Interface::loadControl (
    YAML::Node node ) [protected]
```

Definition at line 134 of file [constructor.cpp](#).

```
00135 {
00136     for(auto stroke : node)
```

```

00137     {
00138         KeyStroke* k = new KeyStroke();
00139         for(auto key : stroke["key"])
00140         {
00141             k->add(toKey(key.as<std::string>()));
00142         }
00143         std::string action = stroke["action"].as<std::string>();
00144
00145         if(action == "move-object")
00146         {
00147             int id = stroke["args"][0].as<int>();
00148             float v = stroke["args"][1].as<float>() / 100.0;
00149             float x = stroke["args"][2].as<float>();
00150             float y = stroke["args"][3].as<float>();
00151             moveObjectAction* action = new moveObjectAction(containers[id], fPoint({x, y}), v);
00152             k->addAction(action);
00153         }
00154
00155         keystrokes.push_back(k);
00156     }
00157 }

```

### 7.12.3.34 loadFocus()

```

void Container::loadFocus (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 218 of file [constructor.cpp](#).

```

00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }

```

### 7.12.3.35 loadName()

```

bool Container::loadName (
    YAML::Node node ) [protected], [inherited]

```

Definition at line 111 of file [constructor.cpp](#).

```

00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }

```

### 7.12.3.36 loadObject()

```

void Interface::loadObject (
    YAML::Node node ) [protected]

```

Definition at line 117 of file [constructor.cpp](#).

```

00118 {
00119     for(auto i : node)
00120     {
00121         Rectangle rel({0, 0, 0, 0});
00122         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00123         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00124         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00125         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00126         Container *obj;
00127         obj = new Object(this, rel);
00128         obj->linkContent(i["path"].as<std::string>());
00129         containers.push_back(obj);
00130     }
00131 }

```

## 7.12.3.37 loadSprites()

```
void Container::loadSprites (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 122 of file [constructor.cpp](#).

```
00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])
00141         {
00142             float x, y, w, h;
00143             int repeat = 1;
00144             int gapX = 0;
00145             int gapY = 0;
00146
00147             int dx = 1;
00148             int dy = 1;
00149
00150             if(img["x"])
00151                 x = img["x"].as<float>() / 100.0;
00152             else x = 0;
00153             if(img["y"])
00154                 y = img["y"].as<float>() / 100.0;
00155             else y = 0;
00156             if(img["w"])
00157                 w = img["w"].as<float>() / 100.0;
00158             else w = 1;
00159             if(img["h"])
00160                 h = img["h"].as<float>() / 100.0;
00161             else h = 1;
00162             if(img["repeat"])
00163                 repeat = img["repeat"].as<int>();
00164             if(img["gapX"])
00165                 gapX = img["gapX"].as<int>();
00166             if(img["gapY"])
00167                 gapY = img["gapY"].as<int>();
00168
00169             if(img["dx"])
00170                 dx = img["dx"].as<int>();
00171             if(dx < 0) dx = -1;
00172             else dx = 1;
00173
00174             if(img["dy"])
00175                 dy = img["dy"].as<int>();
00176             if(dy < 0) dy = -1;
00177             else dy = 1;
00178
00179             int imgw = image.width;
00180             int imgh = image.height;
00181
00182             if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183             {
00184                 for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185                 {
00186                     for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                     {
00188                         Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                         Image img2 = ImageFromImage(image, rect);
00190                         Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                         Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                         sprites.back().push_back(vis);
00193
00194                         UnloadImage(img2);
00195                     }
00196                 }
00197             }else
00198             {
00199                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200                 {
```

```

00201         for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202         {
00203             Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204             Image img2 = ImageFromImage(image, rect);
00205             Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206             Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207             sprites.back().push_back(vis);
00208
00209             UnloadImage(img2);
00210         }
00211     }
00212 }
00213 }
00214     UnloadImage(image);
00215 }
00216 }

```

### 7.12.3.38 moveBy() [1/2]

```

void Frame::moveBy (
    fPoint rel ) [inherited]

```

Definition at line 65 of file [arithmetic.cpp](#).

```

00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }

```

### 7.12.3.39 moveBy() [2/2]

```

void Frame::moveBy (
    int x,
    int y ) [inherited]

```

Definition at line 75 of file [arithmetic.cpp](#).

```

00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }

```

### 7.12.3.40 moveCenterTo() [1/2]

```

void Frame::moveCenterTo (
    fPoint rel ) [inherited]

```

Definition at line 43 of file [arithmetic.cpp](#).

```

00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }

```

**7.12.3.41 moveCenterTo() [2/2]**

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

**7.12.3.42 moveTo() [1/2]**

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

**7.12.3.43 moveTo() [2/2]**

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.12.3.44 nextImage()**

```
void Container::nextImage ( ) [inherited]
```

move to next state of the sprite

Definition at line 247 of file [constructor.cpp](#).

```
00248 {
00249     if(sprites.empty()) return;
00250     focus[1]++;
00251     if(focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
```



### 7.12.3.45 nextSprite()

void Container::nextSprite ( ) [inherited]

move to the next sprite

Definition at line 261 of file [constructor.cpp](#).

```
00262 {
00263     if (sprites.empty()) return;
00264     focus[0]++;
00265     if (focus[0] >= sprites.size()) focus[0] = 0;
00266 }
```

### 7.12.3.46 operator fRect()

Frame::operator fRect ( ) const [inherited]

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

### 7.12.3.47 operator iRect()

Frame::operator iRect ( ) const [inherited]

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

### 7.12.3.48 operator Rectangle()

Frame::operator Rectangle ( ) const [inherited]

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

### 7.12.3.49 OverlappingArea() [1/2]

float Container::OverlappingArea (   
 Container \* container ) [inherited]

Definition at line 34 of file [overlap.cpp](#).

```
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if (w < 0 || h < 0) return 0;
00043     return w * h;
00044 }
```

**7.12.3.50 OverlappingArea() [2/2]**

```
float Container::OverlappingArea (
    Rectangle rec ) [inherited]
```

Definition at line 23 of file [overlap.cpp](#).

```
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
```

**7.12.3.51 plug() [1/2]**

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position

**Parameters**

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
```

**7.12.3.52 plug() [2/2]**

```
void Frame::plug (
    Frame * par,
    fRect rel ) [inherited]
```

attach a frame to a parent by relative position

**Parameters**

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 12 of file [family.cpp](#).

```
00013 {
00014     if(par == nullptr)
00015     {
```

```

00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }

```

### 7.12.3.53 prevImage()

void Container::prevImage ( ) [inherited]

move to previous state of the sprite

Definition at line 254 of file [constructor.cpp](#).

```

00255 {
00256     if(sprites.empty()) return;
00257     focus[1]--;
00258     if(focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }

```

### 7.12.3.54 prevSprite()

void Container::prevSprite ( ) [inherited]

move to the previous sprite

Definition at line 268 of file [constructor.cpp](#).

```

00269 {
00270     if(sprites.empty()) return;
00271     focus[0]--;
00272     if(focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }

```

### 7.12.3.55 react()

Action \* Interface::react ( ) [override], [virtual]

Reimplemented from [Container](#).

Definition at line 38 of file [action.cpp](#).

```

00039 {
00040     if(!isVisible()) return nullptr;
00041     PacketAction* packet = nullptr;
00042
00043     Action* action = Container::react();
00044
00045     if(action != nullptr)
00046     {
00047         packet = new PacketAction();
00048         packet->addAction(action);
00049     }
00050
00051     for(auto i : keystrokes)
00052     {
00053         Action* action = i->react();
00054         if(action != nullptr)
00055         {
00056             if(packet == nullptr) packet = new PacketAction();
00057             packet->addAction(action);
00058         }
00059     }
00060
00061     for(auto i : containers)
00062     {
00063         Action* action = i->react();
00064         if(action != nullptr)
00065         {
00066             if(packet == nullptr) packet = new PacketAction();
00067             packet->addAction(action);
00068         }
00069     }
00070
00071     return packet;
00072 }

```

**7.12.3.56 removeSubframe()**

```
void Frame::removeSubframe (
    Frame * subframe ) [protected], [inherited]
```

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

**Parameters**

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

**7.12.3.57 resize() [1/2]**

```
void Frame::resize (
    fPoint rel ) [inherited]
```

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

**7.12.3.58 resize() [2/2]**

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

**7.12.3.59 setProbability()**

```
void Container::setProbability (
    int prob ) [inherited]
```

Definition at line 280 of file [constructor.cpp](#).

```
00281 {
00282     probability = prob;
00283 }
```

**7.12.3.60 setRelative()**

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

**7.12.3.61 show()**

```
void Container::show ( ) [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     visible = true;
00014 }
```

**7.12.3.62 toggleVisibility()**

```
void Container::toggleVisibility ( ) [inherited]
```

Definition at line 21 of file [arithmetic.cpp](#).

```
00022 {
00023     visible = !visible;
00024 }
```

**7.12.3.63 unplug()**

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

### 7.12.3.64 updateFrame()

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual], [inherited]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
```

## 7.12.4 Friends And Related Symbol Documentation

### 7.12.4.1 moveObjectAction

```
friend class moveObjectAction [friend]
```

Definition at line 23 of file [interface.hpp](#).

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

- [src/interface/include/interface.hpp](#)
- [src/interface/src/action.cpp](#)
- [src/interface/src/arithmetic.cpp](#)
- [src/interface/src/constructor.cpp](#)
- [src/interface/src/destructor.cpp](#)

## 7.13 KeyStroke Class Reference

manages the link between a key and the actions it performs

```
#include <keystroke.hpp>
```

### Public Member Functions

- [KeyStroke](#) ()
- [KeyStroke](#) (std::vector< int >)
- [~KeyStroke](#) ()
- int [size](#) ()
- void [add](#) (unsigned char)
- void [setAction](#) (std::vector< [Action](#) \* >)
- void [addAction](#) ([Action](#) \*)
- void [chooseAction](#) (int)
- int [getCurrent](#) (int)
- void [nextAction](#) ()
- [Action](#) \* [react](#) ()

### 7.13.1 Detailed Description

manages the link between a key and the actions it performs

Definition at line 16 of file [keystroke.hpp](#).

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 KeyStroke() [1/2]

```
KeyStroke::KeyStroke ( )
```

Definition at line 6 of file [keystroke.cpp](#).

```
00007 {  
00008     id = 0;  
00009 }
```

#### 7.13.2.2 KeyStroke() [2/2]

```
KeyStroke::KeyStroke (  
    std::vector< int > k )
```

Definition at line 11 of file [keystroke.cpp](#).

```
00012 {  
00013     key = k;  
00014     id = 0;  
00015 }
```

#### 7.13.2.3 ~KeyStroke()

```
KeyStroke::~~KeyStroke ( )
```

Definition at line 17 of file [keystroke.cpp](#).

```
00018 {  
00019     for(auto &a : action)  
00020     {  
00021         delete a;  
00022     }  
00023 }
```

### 7.13.3 Member Function Documentation

#### 7.13.3.1 add()

```
void KeyStroke::add (  
    unsigned char k )
```

Definition at line 30 of file [keystroke.cpp](#).

```
00031 {  
00032     key.push_back(k);  
00033 }
```

### 7.13.3.2 addAction()

```
void KeyStroke::addAction (
    Action * a )
```

Definition at line 39 of file [keystroke.cpp](#).

```
00040 {
00041     action.push_back(a);
00042 }
```

### 7.13.3.3 chooseAction()

```
void KeyStroke::chooseAction (
    int i )
```

Definition at line 52 of file [keystroke.cpp](#).

```
00053 {
00054     id = i;
00055 }
```

### 7.13.3.4 getCurrent()

```
int KeyStroke::getCurrent (
    int i )
```

Definition at line 57 of file [keystroke.cpp](#).

```
00058 {
00059     return id;
00060 }
```

### 7.13.3.5 nextAction()

```
void KeyStroke::nextAction ( )
```

Definition at line 62 of file [keystroke.cpp](#).

```
00063 {
00064     id = (id + 1) % action.size();
00065 }
```

### 7.13.3.6 react()

```
Action * KeyStroke::react ( )
```

Definition at line 44 of file [keystroke.cpp](#).

```
00045 {
00046     for(auto k : key)
00047     {
00048         if(!IsKeyDown(k)) return nullptr;
00049     }
00050     return action[id]->clone();
00051 }
```



### 7.13.3.7 setAction()

```
void KeyStroke::setAction (
    std::vector< Action * > a )
```

Definition at line 34 of file [keystroke.cpp](#).

```
00035 {
00036     action = a;
00037 }
```

### 7.13.3.8 size()

```
int KeyStroke::size ( )
```

Definition at line 25 of file [keystroke.cpp](#).

```
00026 {
00027     return key.size();
00028 }
```

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

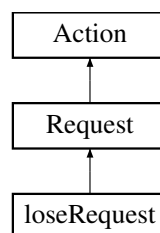
- [src/utls/include/keystroke.hpp](#)
- [src/utls/src/keystroke.cpp](#)

## 7.14 loseRequest Class Reference

request sent when the player loses

```
#include <request.hpp>
```

Inheritance diagram for loseRequest:



### Public Member Functions

- [loseRequest \(\)](#)=default
- [loseRequest \(loseRequest \\*\)](#)
- [~loseRequest \(\)](#)=default
- [int isRequest \(\)](#) override
- [Action \\* clone \(\)](#) override
- [virtual bool isPackage \(\)](#)
- [virtual void execute \(\)](#)
- [virtual std::vector< Action \\* > unpack \(\)](#)
- [virtual ARGS & getArgs \(\)](#)

## Protected Attributes

- [ARGS](#) `args`

### 7.14.1 Detailed Description

request sent when the player loses

Definition at line 44 of file [request.hpp](#).

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 loseRequest() [1/2]

```
loseRequest::loseRequest ( ) [default]
```

#### 7.14.2.2 loseRequest() [2/2]

```
loseRequest::loseRequest (
    loseRequest * other )
```

Definition at line 4 of file [lose.cpp](#).

```
00005 {
00006     args = other->args;
00007 }
```

#### 7.14.2.3 ~loseRequest()

```
loseRequest::~~loseRequest ( ) [default]
```

### 7.14.3 Member Function Documentation

#### 7.14.3.1 clone()

```
Action * loseRequest::clone ( ) [override], [virtual]
```

Reimplemented from [Request](#).

Definition at line 14 of file [lose.cpp](#).

```
00015 {
00016     return new loseRequest(this);
00017 }
```

#### 7.14.3.2 execute()

```
void Action::execute ( ) [virtual], [inherited]
```

Reimplemented in [CloseAction](#), [resizeAction](#), [PacketAction](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 25 of file [action.cpp](#).

```
00026 {
00027 }
```

### 7.14.3.3 getArgs()

[ARGS](#) & Action::getArgs ( ) [virtual], [inherited]

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.14.3.4 isPackage()

bool Action::isPackage ( ) [virtual], [inherited]

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.14.3.5 isRequest()

int loseRequest::isRequest ( ) [override], [virtual]

Reimplemented from [Action](#).

Definition at line 9 of file [lose.cpp](#).

```
00010 {  
00011     return REQUEST::LOSE;  
00012 }
```

### 7.14.3.6 unpack()

std::vector< [Action](#) \* > Action::unpack ( ) [virtual], [inherited]

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

## 7.14.4 Member Data Documentation

### 7.14.4.1 args

[ARGS](#) Request::args [protected], [inherited]

Definition at line 17 of file [request.hpp](#).

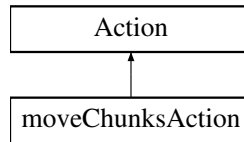
The documentation for this class was generated from the following files:

- [src/action/include/request.hpp](#)
- [src/action/src/request/lose.cpp](#)

## 7.15 moveChunksAction Class Reference

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

Inheritance diagram for moveChunksAction:



### Public Member Functions

- [moveChunksAction](#) ([Game](#) \*, [fPoint](#))
- [moveChunksAction](#) ([Game](#) \*, [fPoint](#), float)
- [~moveChunksAction](#) ()
- void [execute](#) () override
- [Action](#) \* [clone](#) () override
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.15.1 Detailed Description

Definition at line 50 of file [game.hpp](#).

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 moveChunksAction() [1/2]

```
moveChunksAction::moveChunksAction (
    Game * game,
    fPoint delta )
```

Definition at line 4 of file [movechunk.cpp](#).

```
00005 {
00006     this->game = game;
00007     this->delta = delta;
00008 }
```

#### 7.15.2.2 moveChunksAction() [2/2]

```
moveChunksAction::moveChunksAction (
    Game * game,
    fPoint d,
    float v )
```

Definition at line 10 of file [movechunk.cpp](#).

```
00011 {
00012     this->game = game;
00013     this->direction = d;
00014     this->speed = v;
00015
00016     float angle = VECTOR2D::getAngle(direction);
00017     delta[0] = cos(angle) * speed;
00018     delta[1] = sin(angle) * speed;
00019 }
```

### 7.15.2.3 ~moveChunksAction()

`moveChunksAction::~~moveChunksAction ( )`

Definition at line 21 of file [movechunk.cpp](#).

```
00022 {  
00023 }
```

## 7.15.3 Member Function Documentation

### 7.15.3.1 clone()

`Action * moveChunksAction::clone ( ) [override], [virtual]`

Reimplemented from [Action](#).

Definition at line 34 of file [movechunk.cpp](#).

```
00035 {  
00036     return new moveChunksAction(game, delta);  
00037 }
```

### 7.15.3.2 execute()

`void moveChunksAction::execute ( ) [override], [virtual]`

Reimplemented from [Action](#).

Definition at line 25 of file [movechunk.cpp](#).

```
00026 {  
00027     for(auto& chunk : game->chunks)  
00028     {  
00029         chunk->moveBy(delta);  
00030     }  
00031     game->loadMap();  
00032 }
```

### 7.15.3.3 getArgs()

`ARGS & Action::getArgs ( ) [virtual], [inherited]`

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.15.3.4 isPackage()

`bool Action::isPackage ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

## 7.15.3.5 isRequest()

```
int Action::isRequest ( ) [virtual], [inherited]
```

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {
00017     return 0;
00018 }
```

## 7.15.3.6 unpack()

```
std::vector< Action * > Action::unpack ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {
00036     return std::vector<Action*> ({this});
00037 }
```

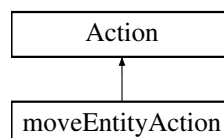
The documentation for this class was generated from the following files:

- [src/game/include/game.hpp](#)
- [src/game/src/action/movechunk.cpp](#)

## 7.16 moveEntityAction Class Reference

```
#include <chunk.hpp>
```

Inheritance diagram for moveEntityAction:



## Public Member Functions

- [moveEntityAction](#) ([Chunk](#) \*)
- [~moveEntityAction](#) ()
- void [execute](#) () override
- [Action](#) \* [clone](#) () override
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.16.1 Detailed Description

Definition at line 53 of file [chunk.hpp](#).

### 7.16.2 Constructor & Destructor Documentation

#### 7.16.2.1 moveEntityAction()

```
moveEntityAction::moveEntityAction (
    Chunk * chunk )
```

Definition at line 3 of file [moveentity.cpp](#).

```
00003                                     : chunk (chunk)
00004 {
00005 }
```

#### 7.16.2.2 ~moveEntityAction()

```
moveEntityAction::~moveEntityAction ( )
```

Definition at line 7 of file [moveentity.cpp](#).

```
00008 {
00009 }
```

### 7.16.3 Member Function Documentation

#### 7.16.3.1 clone()

```
Action * moveEntityAction::clone ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 16 of file [moveentity.cpp](#).

```
00017 {
00018     return new moveEntityAction (chunk);
00019 }
```

#### 7.16.3.2 execute()

```
void moveEntityAction::execute ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 11 of file [moveentity.cpp](#).

```
00012 {
00013     chunk->movingEntity ();
00014 }
```

### 7.16.3.3 getArgs()

`ARGS & Action::getArgs ( ) [virtual], [inherited]`

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.16.3.4 isPackage()

`bool Action::isPackage ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.16.3.5 isRequest()

`int Action::isRequest ( ) [virtual], [inherited]`

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {  
00017     return 0;  
00018 }
```

### 7.16.3.6 unpack()

`std::vector< Action * > Action::unpack ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

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

- [src/chunk/include/chunk.hpp](#)
- [src/chunk/src/action/moveentity.cpp](#)

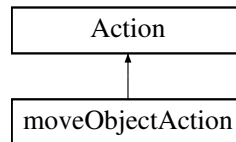


## 7.17 moveObjectAction Class Reference

manages the features of a movement, including which object, speed, direction etc.

```
#include <interface.hpp>
```

Inheritance diagram for moveObjectAction:



### Public Member Functions

- [moveObjectAction](#) ([Container](#) \*obj, [fPoint](#) delta)
- [moveObjectAction](#) ([Container](#) \*obj, [fPoint](#) dir, float speed)
- [~moveObjectAction](#) ()
- void [execute](#) () override
- [Action](#) \* [clone](#) () override
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.17.1 Detailed Description

manages the features of a movement, including which object, speed, direction etc.

Definition at line 60 of file [interface.hpp](#).

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 moveObjectAction() [1/2]

```
moveObjectAction::moveObjectAction (
    Container * obj,
    fPoint delta )
```

Definition at line 4 of file [moveobject.cpp](#).

```
00004                                     : obj(obj), delta(delta)
00005 {
00006 }
```

### 7.17.2.2 moveObjectAction() [2/2]

```
moveObjectAction::moveObjectAction (
    Container * obj,
    fPoint dir,
    float speed )
```

Definition at line 8 of file [moveobject.cpp](#).

```
00008                                     : obj(obj), dir(dir),
    speed(speed)
00009 {
00010     float angle = VECTOR2D::getAngle(dir);
00011     delta[0] = cos(angle) * speed;
00012     delta[1] = sin(angle) * speed;
00013 }
```

### 7.17.2.3 ~moveObjectAction()

```
moveObjectAction::~moveObjectAction ( )
```

Definition at line 15 of file [moveobject.cpp](#).

```
00016 {
00017 }
```

## 7.17.3 Member Function Documentation

### 7.17.3.1 clone()

```
Action * moveObjectAction::clone ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 25 of file [moveobject.cpp](#).

```
00026 {
00027     return new moveObjectAction(obj, delta);
00028 }
```

### 7.17.3.2 execute()

```
void moveObjectAction::execute ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 19 of file [moveobject.cpp](#).

```
00020 {
00021     obj->moveBy(delta);
00022 }
```

### 7.17.3.3 getArgs()

```
ARGS & Action::getArgs ( ) [virtual], [inherited]
```

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {
00041     return NONE_ARGS;
00042 }
```

### 7.17.3.4 isPackage()

```
bool Action::isPackage ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.17.3.5 isRequest()

```
int Action::isRequest ( ) [virtual], [inherited]
```

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {  
00017     return 0;  
00018 }
```

### 7.17.3.6 unpack()

```
std::vector< Action * > Action::unpack ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

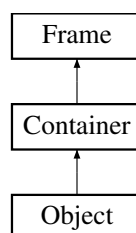
The documentation for this class was generated from the following files:

- [src/interface/include/interface.hpp](#)
- [src/interface/src/action/moveobject.cpp](#)

## 7.18 Object Class Reference

```
#include <object.hpp>
```

Inheritance diagram for Object:



## Public Member Functions

- [Object](#) ([Frame](#) \*, [Rectangle](#))
- [Object](#) ([Object](#) \*)
- [Object](#) ([Object](#) \*, [Rectangle](#))
- [Object](#) ([Object](#) \*, [Frame](#) \*, [Rectangle](#))
- [~Object](#) ()
- virtual std::string [linkContent](#) (std::string) override
- virtual std::string [linkContentAbsolute](#) (std::string) override
- virtual [Action](#) \* [react](#) () override
- void [draw](#) () override
- std::string [getName](#) ()
- void [setProbability](#) (int)
- int [getProbability](#) ()
- void [chooseSprite](#) (int)  
*choose a specific sprite from a vector of sprites*
- void [chooseImage](#) (int)  
*choose the state of the sprite*
- void [chooseImage](#) (int, int)  
*choose the state of the sprite*
- void [nextImage](#) ()  
*move to next state of the sprite*
- void [prevImage](#) ()  
*move to previous state of the sprite*
- void [nextSprite](#) ()  
*move to the next sprite*
- void [prevSprite](#) ()  
*move to the previous sprite*
- bool [isOverlapping](#) ([fPoint](#))
- bool [isOverlapping](#) ([Rectangle](#))
- bool [isOverlapping](#) ([Container](#) \*)
- float [OverlappingArea](#) ([Rectangle](#))
- float [OverlappingArea](#) ([Container](#) \*)
- void [show](#) ()
- void [hide](#) ()
- void [toggleVisibility](#) ()
- bool [isVisible](#) ()
- int [getInstanceId](#) ()
- virtual [Action](#) \* [getRuntimeEvent](#) ()
- void [plug](#) ([Frame](#) \*par, [fRect](#) rel)  
*attach a frame to a parent by relative position*
- void [plug](#) ([Frame](#) \*par)  
*attach a frame to a parent by old relative position*
- void [unplug](#) ()  
*detach a frame from its parent*
- void [moveTo](#) ([fPoint](#) rel)
- void [moveTo](#) (int x, int y)
- void [moveCenterTo](#) ([fPoint](#) rel)
- void [moveCenterTo](#) (int x, int y)
- void [moveBy](#) ([fPoint](#) rel)
- void [moveBy](#) (int, int)
- void [resize](#) ([fPoint](#) rel)
- void [resize](#) (int w, int h)

- const Rectangle & [getFrame](#) () const
- const [fRect](#) & [getRelative](#) () const
- [Frame](#) \* [getParent](#) ()
- void [setRelative](#) ([fRect](#) rel)
- const [fPoint](#) & [getCenter](#) () const
- const float & [getX](#) () const
- const float & [getY](#) () const
- const float & [getW](#) () const
- const float & [getH](#) () const
- [operator Rectangle](#) () const
- [operator fRect](#) () const
- [operator iRect](#) () const

### Protected Member Functions

- void [loadControl](#) (YAML::Node node)
- bool [loadName](#) (YAML::Node node)
- void [loadSprites](#) (YAML::Node node)
- void [loadFocus](#) (YAML::Node node)
- virtual void [updateFrame](#) (bool recursive=false)
- bool [isroot](#) () const  
*return true if this frame is root*
- void [addSubframe](#) ([Frame](#) \*subframe)  
*Add a subframe to this frame.*
- void [removeSubframe](#) ([Frame](#) \*subframe)  
*Remove a subframe from this frame.*
- void [beginUpdate](#) ()
- void [endUpdate](#) ()

## 7.18.1 Detailed Description

Definition at line 8 of file [object.hpp](#).

## 7.18.2 Constructor & Destructor Documentation

### 7.18.2.1 Object() [1/4]

```
Object::Object (
    Frame * f,
    Rectangle rel )
```

Definition at line 6 of file [constructor.cpp](#).

```
00006                                     : Container(f, rel)
00007 {
00008     waitUntil = std::chrono::steady_clock::now();
00009 }
```

### 7.18.2.2 Object() [2/4]

```
Object::Object (
    Object * other )
```

Definition at line 11 of file [constructor.cpp](#).

```
00011         : Container(other)
00012 {
00013     waitUntil = std::chrono::steady_clock::now();
00014 }
```

### 7.18.2.3 Object() [3/4]

```
Object::Object (
    Object * other,
    Rectangle rel )
```

Definition at line 16 of file [constructor.cpp](#).

```
00016         : Container(other, rel)
00017 {
00018     waitUntil = std::chrono::steady_clock::now();
00019 }
```

### 7.18.2.4 Object() [4/4]

```
Object::Object (
    Object * other,
    Frame * f,
    Rectangle rel )
```

Definition at line 21 of file [constructor.cpp](#).

```
00021         : Container(other, f, rel)
00022 {
00023     waitUntil = std::chrono::steady_clock::now();
00024 }
```

### 7.18.2.5 ~Object()

```
Object::~Object ( )
```

Definition at line 4 of file [destructor.cpp](#).

```
00005 {
00006     for (auto &stroke : strokes)
00007     {
00008         delete stroke.stroke;
00009     }
00010     strokes.clear();
00011 }
00012 }
```

## 7.18.3 Member Function Documentation

### 7.18.3.1 addSubframe()

```
void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]
```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

## Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```
00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }
```

**7.18.3.2 beginUpdate()**

```
void Frame::beginUpdate ( ) [protected], [inherited]
```

Definition at line 113 of file [family.cpp](#).

```
00114 {
00115     mtx.lock();
00116 }
```

**7.18.3.3 chooseImage() [1/2]**

```
void Container::chooseImage (
    int index ) [inherited]
```

choose the state of the sprite

Definition at line 231 of file [constructor.cpp](#).

```
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
```

**7.18.3.4 chooseImage() [2/2]**

```
void Container::chooseImage (
    int index,
    int index2 ) [inherited]
```

choose the state of the sprite

Definition at line 238 of file [constructor.cpp](#).

```
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
```

**7.18.3.5 chooseSprite()**

```
void Container::chooseSprite (
    int index ) [inherited]
```

choose a specific sprite from a vector of sprites

Definition at line 224 of file [constructor.cpp](#).

```
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
```

### 7.18.3.6 draw()

```
void Object::draw ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 19 of file [arithmetic.cpp](#).

```
00020 {  
00021     Container::draw\(\);  
00022     return ;  
00023 }
```

### 7.18.3.7 endUpdate()

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```
00119 {  
00120     mtx.unlock();  
00121 }
```

### 7.18.3.8 getCenter()

```
const fPoint & Frame::getCenter ( ) const [inherited]
```

Definition at line 131 of file [arithmetic.cpp](#).

```
00132 {  
00133     std::lock_guard<std::mutex> lock(mtx);  
00134     static fPoint resu;  
00135     if(isroot())  
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};  
00137     else  
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};  
00139  
00140     return resu;  
00141 }
```

### 7.18.3.9 getFrame()

```
const Rectangle & Frame::getFrame ( ) const [inherited]
```

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {  
00107     std::lock_guard<std::mutex> lock(mtx);  
00108     return frame;  
00109 }
```

### 7.18.3.10 getH()

```
const float & Frame::getH ( ) const [inherited]
```

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```



#### 7.18.3.11 getInstanceId()

```
int Container::getInstanceId ( ) [inherited]
```

Definition at line 31 of file [arithmetic.cpp](#).

```
00032 {  
00033     return instance_id;  
00034 }
```

#### 7.18.3.12 getName()

```
std::string Container::getName ( ) [inherited]
```

Definition at line 275 of file [constructor.cpp](#).

```
00276 {  
00277     return name;  
00278 }
```

#### 7.18.3.13 getParent()

```
Frame * Frame::getParent ( ) [inherited]
```

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```

#### 7.18.3.14 getProbability()

```
int Container::getProbability ( ) [inherited]
```

Definition at line 285 of file [constructor.cpp](#).

```
00286 {  
00287     return probability;  
00288 }
```

#### 7.18.3.15 getRelative()

```
const fRect & Frame::getRelative ( ) const [inherited]
```

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {  
00113     std::lock_guard<std::mutex> lock(mtx);  
00114     return relative;  
00115 }
```

#### 7.18.3.16 getRuntimeEvent()

```
Action * Container::getRuntimeEvent ( ) [virtual], [inherited]
```

Reimplemented in [Chunk](#), [Game](#), and [Interface](#).

Definition at line 41 of file [arithmetic.cpp](#).

```
00042 {  
00043     return nullptr;  
00044 }
```

### 7.18.3.17 getW()

```
const float & Frame::getW ( ) const [inherited]
```

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {  
00157     std::lock_guard<std::mutex> lock(mtx);  
00158     return frame.width;  
00159 }
```

### 7.18.3.18 getX()

```
const float & Frame::getX ( ) const [inherited]
```

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {  
00145     std::lock_guard<std::mutex> lock(mtx);  
00146     return frame.x;  
00147 }
```

### 7.18.3.19 getY()

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {  
00151     std::lock_guard<std::mutex> lock(mtx);  
00152     return frame.y;  
00153 }
```

### 7.18.3.20 hide()

```
void Container::hide ( ) [inherited]
```

Definition at line 16 of file [arithmetic.cpp](#).

```
00017 {  
00018     visible = false;  
00019 }
```

### 7.18.3.21 isOverlapping() [1/3]

```
bool Container::isOverlapping (  
    Container * container ) [inherited]
```

Definition at line 16 of file [overlap.cpp](#).

```
00017 {  
00018     Rectangle rec = getFrame();  
00019     Rectangle rec2 = container->getFrame();  
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +  
        rec2.height && rec.y + rec.height >= rec2.y);  
00021 }
```

**7.18.3.22 isOverlapping() [2/3]**

```
bool Container::isOverlapping (
    fPoint point ) [inherited]
```

Definition at line 3 of file [overlap.cpp](#).

```
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
        rec.y + rec.height);
00007
00008 }
```

**7.18.3.23 isOverlapping() [3/3]**

```
bool Container::isOverlapping (
    Rectangle rec ) [inherited]
```

Definition at line 10 of file [overlap.cpp](#).

```
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
        rec2.height && rec.y + rec.height >= rec2.y);
00014 }
```

**7.18.3.24 isroot()**

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

**7.18.3.25 isVisible()**

```
bool Container::isVisible ( ) [inherited]
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     return visible;
00029 }
```

**7.18.3.26 linkContent()**

```
std::string Object::linkContent (
    std::string path ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 26 of file [constructor.cpp](#).

```
00027 {
00028     return linkContentAbsolute(PATB::OBJECT_ + path);
00029 }
```

**7.18.3.27 linkContentAbsolute()**

```
std::string Object::linkContentAbsolute (
    std::string path ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 31 of file [constructor.cpp](#).

```
00032 {
00033     YAML::Node node = YAML_FILE::readFile(path);
00034     if(!loadName(node)) return "";
00035     if(node["textures"]) loadSprites(node["textures"]);
00036     if(node["control"]) loadControl(node["control"]);
00037
00038     chooseImage(0, 0);
00039     if(node["focus"])
00040         loadFocus(node["focus"]);
00041
00042     return "";
00043 }
```

**7.18.3.28 loadControl()**

```
void Object::loadControl (
    YAML::Node node ) [protected]
```

Definition at line 45 of file [constructor.cpp](#).

```
00046 {
00047     for(auto stroke : node)
00048     {
00049         strokes.emplace_back();
00050         KeyStroke* k = new KeyStroke();
00051         for(auto key : stroke["key"])
00052         {
00053             k->add(toKey(key.as<std::string>()));
00054         }
00055         for(auto sprite : stroke["sprite"])
00056         {
00057             iPoint p;
00058             int delay = 0;
00059             p[0] = sprite[0].as<int>();
00060             p[1] = sprite[1].as<int>();
00061             if(p.size() >= 3)
00062                 delay = sprite[2].as<int>();
00063             k->addAction(new changeImageAction(this, p));
00064         }
00065         strokes.back().stroke = k;
00066     }
00067 }
```

**7.18.3.29 loadFocus()**

```
void Container::loadFocus (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 218 of file [constructor.cpp](#).

```
00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }
```

### 7.18.3.30 loadName()

```
bool Container::loadName (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 111 of file [constructor.cpp](#).

```
00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }
```

### 7.18.3.31 loadSprites()

```
void Container::loadSprites (
    YAML::Node node ) [protected], [inherited]
```

Definition at line 122 of file [constructor.cpp](#).

```
00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])
00141         {
00142             float x, y, w, h;
00143             int repeat = 1;
00144             int gapX = 0;
00145             int gapY = 0;
00146
00147             int dx = 1;
00148             int dy = 1;
00149
00150             if(img["x"])
00151                 x = img["x"].as<float>() / 100.0;
00152             else x = 0;
00153             if(img["y"])
00154                 y = img["y"].as<float>() / 100.0;
00155             else y = 0;
00156             if(img["w"])
00157                 w = img["w"].as<float>() / 100.0;
00158             else w = 1;
00159             if(img["h"])
00160                 h = img["h"].as<float>() / 100.0;
00161             else h = 1;
00162             if(img["repeat"])
00163                 repeat = img["repeat"].as<int>();
00164             if(img["gapX"])
00165                 gapX = img["gapX"].as<int>();
00166             if(img["gapY"])
00167                 gapY = img["gapY"].as<int>();
00168
00169             if(img["dx"])
00170                 dx = img["dx"].as<int>();
00171             if(dx < 0) dx = -1;
00172             else dx = 1;
00173
00174             if(img["dy"])
00175                 dy = img["dy"].as<int>();
00176             if(dy < 0) dy = -1;
00177             else dy = 1;
00178 }
```

```

00179         int imgw = image.width;
00180         int imgh = image.height;
00181
00182         if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183         {
00184             for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185             {
00186                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                 {
00188                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                     Image img2 = ImageFromImage(image, rect);
00190                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                     sprites.back().push_back(vis);
00193
00194                     UnloadImage(img2);
00195                 }
00196             }
00197         }else
00198         {
00199             for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200             {
00201                 for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202                 {
00203                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204                     Image img2 = ImageFromImage(image, rect);
00205                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207                     sprites.back().push_back(vis);
00208
00209                     UnloadImage(img2);
00210                 }
00211             }
00212         }
00213     }
00214     UnloadImage(image);
00215 }
00216 }

```

### 7.18.3.32 moveBy() [1/2]

```

void Frame::moveBy (
    fPoint rel ) [inherited]

```

Definition at line 65 of file [arithmetic.cpp](#).

```

00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }

```

### 7.18.3.33 moveBy() [2/2]

```

void Frame::moveBy (
    int x,
    int y ) [inherited]

```

Definition at line 75 of file [arithmetic.cpp](#).

```

00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }

```

### 7.18.3.34 moveCenterTo() [1/2]

```
void Frame::moveCenterTo (
    fPoint rel ) [inherited]
```

Definition at line 43 of file [arithmetic.cpp](#).

```
00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }
```

### 7.18.3.35 moveCenterTo() [2/2]

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

### 7.18.3.36 moveTo() [1/2]

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

### 7.18.3.37 moveTo() [2/2]

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.18.3.38 nextImage()**

```
void Container::nextImage ( ) [inherited]
```

move to next state of the sprite

Definition at line 247 of file [constructor.cpp](#).

```
00248 {
00249     if (sprites.empty()) return;
00250     focus[1]++;
00251     if (focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
```

**7.18.3.39 nextSprite()**

```
void Container::nextSprite ( ) [inherited]
```

move to the next sprite

Definition at line 261 of file [constructor.cpp](#).

```
00262 {
00263     if (sprites.empty()) return;
00264     focus[0]++;
00265     if (focus[0] >= sprites.size()) focus[0] = 0;
00266 }
```

**7.18.3.40 operator fRect()**

```
Frame::operator fRect ( ) const [inherited]
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

**7.18.3.41 operator iRect()**

```
Frame::operator iRect ( ) const [inherited]
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

**7.18.3.42 operator Rectangle()**

```
Frame::operator Rectangle ( ) const [inherited]
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```



**7.18.3.43 OverlappingArea()** [1/2]

```
float Container::OverlappingArea (
    Container * container ) [inherited]
```

Definition at line 34 of file [overlap.cpp](#).

```
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if(w < 0 || h < 0) return 0;
00043     return w * h;
00044 }
```

**7.18.3.44 OverlappingArea()** [2/2]

```
float Container::OverlappingArea (
    Rectangle rec ) [inherited]
```

Definition at line 23 of file [overlap.cpp](#).

```
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
```

**7.18.3.45 plug()** [1/2]

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position

**Parameters**

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
```

**7.18.3.46 plug()** [2/2]

```
void Frame::plug (
```

```

    Frame * par,
    fRect rel ) [inherited]

```

attach a frame to a parent by relative position

#### Parameters

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)

Definition at line 12 of file [family.cpp](#).

```

00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }

```

#### 7.18.3.47 prevImage()

```
void Container::prevImage ( ) [inherited]
```

move to previous state of the sprite

Definition at line 254 of file [constructor.cpp](#).

```

00255 {
00256     if(sprites.empty()) return;
00257     focus[1]--;
00258     if(focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }

```

#### 7.18.3.48 prevSprite()

```
void Container::prevSprite ( ) [inherited]
```

move to the previous sprite

Definition at line 268 of file [constructor.cpp](#).

```

00269 {
00270     if(sprites.empty()) return;
00271     focus[0]--;
00272     if(focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }

```

#### 7.18.3.49 react()

```
Action * Object::react ( ) [override], [virtual]
```

Reimplemented from [Container](#).

Definition at line 3 of file [arithmetic.cpp](#).

```

00004 {
00005     if(std::chrono::steady_clock::now() < waitUntil)
00006         return nullptr;
00007     for(int i = 0; i < strokes.size(); i++)
00008     {
00009         Action* a = strokes[i].stroke->react();
00010         if(a == nullptr) continue;
00011         else strokes[i].stroke->nextAction();
00012         return a;
00013     }
00014
00015
00016     return nullptr;
00017 }

```

### 7.18.3.50 removeSubframe()

```
void Frame::removeSubframe (
    Frame * subframe ) [protected], [inherited]
```

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

### 7.18.3.51 resize() [1/2]

```
void Frame::resize (
    fPoint rel ) [inherited]
```

Definition at line 85 of file [arithmetic.cpp](#).

```
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
```

### 7.18.3.52 resize() [2/2]

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

### 7.18.3.53 setProbability()

```
void Container::setProbability (
    int prob ) [inherited]
```

Definition at line 280 of file [constructor.cpp](#).

```
00281 {
00282     probability = prob;
00283 }
```

### 7.18.3.54 setRelative()

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

### 7.18.3.55 show()

```
void Container::show ( ) [inherited]
```

Definition at line 11 of file [arithmetic.cpp](#).

```
00012 {
00013     visible = true;
00014 }
```

### 7.18.3.56 toggleVisibility()

```
void Container::toggleVisibility ( ) [inherited]
```

Definition at line 21 of file [arithmetic.cpp](#).

```
00022 {
00023     visible = !visible;
00024 }
```

### 7.18.3.57 unplug()

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

### 7.18.3.58 updateFrame()

```
void Frame::updateFrame (
    bool recursive = false ) [protected], [virtual], [inherited]
```

Reimplemented in [Visual](#).

Definition at line 3 of file [arithmetic.cpp](#).

```
00004 {
00005
00006     if (parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if (recursive)
00016     {
00017         for (auto& subframe : subframes)
00018         {
00019             subframe->updateFrame (true);
00020         }
00021     }
00022 }
```

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

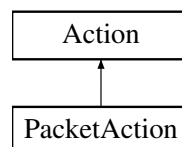
- [src/object/include/object.hpp](#)
- [src/object/src/arithmetic.cpp](#)
- [src/object/src/constructor.cpp](#)
- [src/object/src/destructor.cpp](#)

## 7.19 PacketAction Class Reference

organize selected actions into a package

```
#include <action.hpp>
```

Inheritance diagram for PacketAction:



### Public Member Functions

- [PacketAction](#) ()
- [PacketAction](#) ([PacketAction](#) \*)
- [~PacketAction](#) ()
- bool [isPackage](#) () override
- void [addAction](#) ([Action](#) \*)
- void [addAction](#) ([PacketAction](#) \*)
- std::vector< [Action](#) \* > [unpack](#) () override
- void [execute](#) () override
- [PacketAction](#) \* [clone](#) () override
- virtual int [isRequest](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.19.1 Detailed Description

organize selected actions into a package

Definition at line 52 of file [action.hpp](#).

### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 PacketAction() [1/2]

```
PacketAction::PacketAction ( )
```

Definition at line 44 of file [action.cpp](#).

```
00044         : Action()
00045 {
00046 }
```

#### 7.19.2.2 PacketAction() [2/2]

```
PacketAction::PacketAction (
    PacketAction * action )
```

Definition at line 49 of file [action.cpp](#).

```
00049         : Action(action)
00050 {
00051     for(Action* a : action->actions)
00052         actions.push_back(a->clone());
00053 }
```

#### 7.19.2.3 ~PacketAction()

```
PacketAction::~~PacketAction ( )
```

Definition at line 55 of file [action.cpp](#).

```
00056 {
00057     for(Action* a : actions)
00058         delete a;
00059     actions.clear();
00060 }
```

### 7.19.3 Member Function Documentation

#### 7.19.3.1 addAction() [1/2]

```
void PacketAction::addAction (
    Action * action )
```

Definition at line 67 of file [action.cpp](#).

```
00068 {
00069     actions.push_back(action);
00070 }
```

### 7.19.3.2 addAction() [2/2]

```
void PacketAction::addAction (
    PacketAction * action )
```

Definition at line 72 of file [action.cpp](#).

```
00073 {
00074     for(auto i : action->actions)
00075         actions.push_back(i);
00076     action->actions.clear();
00077 }
```

### 7.19.3.3 clone()

```
PacketAction * PacketAction::clone ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 116 of file [action.cpp](#).

```
00117 {
00118     return new PacketAction(this);
00119 }
```

### 7.19.3.4 execute()

```
void PacketAction::execute ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 107 of file [action.cpp](#).

```
00108 {
00109     for(Action* a : actions)
00110     {
00111         a->execute();
00112     }
00113 }
```

### 7.19.3.5 getArgs()

```
ARGS & Action::getArgs ( ) [virtual], [inherited]
```

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {
00041     return NONE_ARGS;
00042 }
```

### 7.19.3.6 isPackage()

```
bool PacketAction::isPackage ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 62 of file [action.cpp](#).

```
00063 {
00064     return true;
00065 }
```

### 7.19.3.7 isRequest()

```
int Action::isRequest ( ) [virtual], [inherited]
```

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {
00017     return 0;
00018 }
```

### 7.19.3.8 unpack()

```
std::vector< Action * > PacketAction::unpack ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 79 of file [action.cpp](#).

```
00080 {
00081     std::vector<Action*> unpacked;
00082     std::queue<PacketAction*> q;
00083
00084     q.push(this);
00085
00086     while(!q.empty())
00087     {
00088         PacketAction* p = q.front();
00089         q.pop();
00090
00091         for(Action* a : p->actions)
00092         {
00093             if(a->isPackage())
00094             {
00095                 q.push((PacketAction*)a);
00096             }
00097             else
00098             {
00099                 unpacked.push_back(a);
00100             }
00101         }
00102         p->actions.clear();
00103     }
00104     return unpacked;
00105 }
```

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

- [src/action/include/action.hpp](#)
- [src/action/src/action.cpp](#)

## 7.20 RandomEngine Class Reference

```
#include <random.hpp>
```

### Public Member Functions

- [RandomEngine](#) ()
- [RandomEngine](#) (unsigned int seed)
- [~RandomEngine](#) ()
- int [randInt](#) (int min=0, int max=1)
- double [randDouble](#) (double min=0, double max=1)
- char [randChar](#) (char min=0, char max=127)
- std::string [randString](#) (int length, char min, char max)
- std::string [randInt2String](#) (int length, int min=0, int max=9)
- std::string [randString](#) (int length, bool haveDigit=true, bool haveLower=true, bool haveUpper=true, bool haveSpecial=true)



## 7.20.1 Detailed Description

Definition at line 7 of file [random.hpp](#).

## 7.20.2 Constructor & Destructor Documentation

### 7.20.2.1 RandomEngine() [1/2]

```
RandomEngine::RandomEngine ( )
```

Definition at line 6 of file [random.cpp](#).

```
00007 {  
00008     unsigned int seed = std::chrono::system_clock::now().time_since_epoch().count();  
00009     engine.seed(seed);  
00010 }
```

### 7.20.2.2 RandomEngine() [2/2]

```
RandomEngine::RandomEngine (  
    unsigned int seed )
```

Definition at line 12 of file [random.cpp](#).

```
00013 {  
00014     engine.seed(seed);  
00015 }
```

### 7.20.2.3 ~RandomEngine()

```
RandomEngine::~RandomEngine ( )
```

Definition at line 17 of file [random.cpp](#).

```
00018 {  
00019 }
```

## 7.20.3 Member Function Documentation

### 7.20.3.1 randChar()

```
char RandomEngine::randChar (  
    char min = 0,  
    char max = 127 )
```

Definition at line 33 of file [random.cpp](#).

```
00034 {  
00035     std::uniform_int_distribution<int> distribution(min, max);  
00036     return distribution(engine);  
00037 }
```

### 7.20.3.2 randDouble()

```
double RandomEngine::randDouble (
    double min = 0,
    double max = 1 )
```

Definition at line 27 of file [random.cpp](#).

```
00028 {
00029     std::uniform_real_distribution<double> distribution(min, max);
00030     return distribution(engine);
00031 }
```

### 7.20.3.3 randInt()

```
int RandomEngine::randInt (
    int min = 0,
    int max = 1 )
```

Definition at line 21 of file [random.cpp](#).

```
00022 {
00023     std::uniform_int_distribution<int> distribution(min, max);
00024     return distribution(engine);
00025 }
```

### 7.20.3.4 randInt2String()

```
std::string RandomEngine::randInt2String (
    int length,
    int min = 0,
    int max = 9 )
```

Definition at line 49 of file [random.cpp](#).

```
00050 {
00051     std::string str;
00052     for (int i = 0; i < length; i++)
00053     {
00054         str += std::to_string(randInt(min, max));
00055     }
00056     return str;
00057 }
```

### 7.20.3.5 randString() [1/2]

```
std::string RandomEngine::randString (
    int length,
    bool haveDigit = true,
    bool haveLower = true,
    bool haveUpper = true,
    bool haveSpecial = true )
```

Definition at line 59 of file [random.cpp](#).

```
00060 {
00061     std::string str;
00062     std::string digit = "0123456789";
00063     std::string lower = "abcdefghijklmnopqrstuvwxyz";
00064     std::string upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
00065     std::string special = "!@#%^&*()_+{|:;<>?~\`-=[\]\|;'\",./\"";
00066     std::string all = digit + lower + upper + special;
00067     if (haveDigit)
00068     {
00069         str += digit;
00070     }
```



**Public Member Functions**

- [Request](#) ()
- [Request](#) ([Request](#) \*)
- [~Request](#) ()=default
- int [isRequest](#) () override
- virtual [Action](#) \* [clone](#) () override
- virtual bool [isPackage](#) ()
- virtual void [execute](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

**Protected Attributes**

- [ARGS](#) args

**7.21.1 Detailed Description**

sends information to a higher, relevant entity

upgrades information to higher level

Definition at line 14 of file [request.hpp](#).

**7.21.2 Constructor & Destructor Documentation****7.21.2.1 Request() [1/2]**

```
Request::Request ( )
```

Definition at line 4 of file [request.cpp](#).

```
00004         : Action()
00005 {
00006 }
```

**7.21.2.2 Request() [2/2]**

```
Request::Request (
    Request * request )
```

Definition at line 8 of file [request.cpp](#).

```
00008         : Action(request)
00009 {
00010 }
```

**7.21.2.3 ~Request()**

```
Request::~Request ( ) [default]
```

## 7.21.3 Member Function Documentation

### 7.21.3.1 clone()

```
Action * Request::clone ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Reimplemented in [changeInfRequest](#), and [loseRequest](#).

Definition at line 17 of file [request.cpp](#).

```
00018 {  
00019     return new Request(this);  
00020 }
```

### 7.21.3.2 execute()

```
void Action::execute ( ) [virtual], [inherited]
```

Reimplemented in [CloseAction](#), [resizeAction](#), [PacketAction](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 25 of file [action.cpp](#).

```
00026 {  
00027 }
```

### 7.21.3.3 getArgs()

```
ARGS & Action::getArgs ( ) [virtual], [inherited]
```

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.21.3.4 isPackage()

```
bool Action::isPackage ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.21.3.5 isRequest()

```
int Request::isRequest ( ) [override], [virtual]
```

Reimplemented from [Action](#).

Definition at line 12 of file [request.cpp](#).

```
00013 {  
00014     return 1;  
00015 }
```

### 7.21.3.6 `unpack()`

```
std::vector< Action * > Action::unpack ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {
00036     return std::vector<Action*> ({this});
00037 }
```

## 7.21.4 Member Data Documentation

### 7.21.4.1 `args`

[ARGS](#) `Request::args` [protected]

Definition at line 17 of file [request.hpp](#).

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

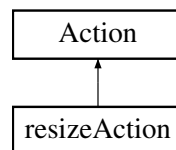
- [src/action/include/request.hpp](#)
- [src/action/src/request.cpp](#)

## 7.22 `resizeAction` Class Reference

manages the resizing of the window

```
#include <window.hpp>
```

Inheritance diagram for `resizeAction`:



### Public Member Functions

- [resizeAction](#) ([Window](#) \*window, float w, float h)
- [~resizeAction](#) ()=default
- void [execute](#) ()
- virtual int [isRequest](#) ()
- virtual bool [isPackage](#) ()
- virtual [Action](#) \* [clone](#) ()
- virtual std::vector< [Action](#) \* > [unpack](#) ()
- virtual [ARGS](#) & [getArgs](#) ()

### 7.22.1 Detailed Description

manages the resizing of the window

Definition at line 197 of file [window.hpp](#).

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 `resizeAction()`

```
resizeAction::resizeAction (
    Window * window,
    float w,
    float h )
```

Definition at line 3 of file [resize.cpp](#).

```
00004 {
00005     win = window;
00006     w = x;
00007     h = y;
00008 }
```

#### 7.22.2.2 `~resizeAction()`

```
resizeAction::~resizeAction ( ) [default]
```

### 7.22.3 Member Function Documentation

#### 7.22.3.1 `clone()`

```
Action * Action::clone ( ) [virtual], [inherited]
```

Reimplemented in [PacketAction](#), [Request](#), [changeInfRequest](#), [loseRequest](#), [moveEntityAction](#), [changeImageAction](#), [moveChunksAction](#), and [moveObjectAction](#).

Definition at line 29 of file [action.cpp](#).

```
00030 {
00031     return this;
00032 }
```

#### 7.22.3.2 `execute()`

```
void resizeAction::execute ( ) [virtual]
```

Reimplemented from [Action](#).

Definition at line 10 of file [resize.cpp](#).

```
00011 {
00012     win->UI.resize(w, h);
00013 }
```

### 7.22.3.3 `getArgs()`

`ARGS & Action::getArgs ( ) [virtual], [inherited]`

Reimplemented in [changeInfRequest](#).

Definition at line 39 of file [action.cpp](#).

```
00040 {  
00041     return NONE_ARGS;  
00042 }
```

### 7.22.3.4 `isPackage()`

`bool Action::isPackage ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#).

Definition at line 20 of file [action.cpp](#).

```
00021 {  
00022     return false;  
00023 }
```

### 7.22.3.5 `isRequest()`

`int Action::isRequest ( ) [virtual], [inherited]`

Reimplemented in [Request](#), [changeInfRequest](#), and [loseRequest](#).

Definition at line 15 of file [action.cpp](#).

```
00016 {  
00017     return 0;  
00018 }
```

### 7.22.3.6 `unpack()`

`std::vector< Action * > Action::unpack ( ) [virtual], [inherited]`

Reimplemented in [PacketAction](#).

Definition at line 34 of file [action.cpp](#).

```
00035 {  
00036     return std::vector<Action*> ({this});  
00037 }
```

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

- [src/window/include/window.hpp](#)
- [src/window/src/action/resize.cpp](#)

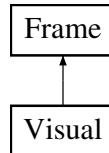


## 7.23 Visual Class Reference

images displayed on screen

```
#include <visual.hpp>
```

Inheritance diagram for Visual:



### Public Member Functions

- [Visual](#) (Texture2D \*, [Frame](#) \*, Rectangle)
- [Visual](#) ([Visual](#) \*)
- [Visual](#) ([Visual](#) \*, Rectangle)
- [Visual](#) ([Visual](#) \*, [Frame](#) \*, Rectangle)
- [~Visual](#) ()
- void [resize](#) ([fPoint](#))
- void [draw](#) ()
- void [plug](#) ([Frame](#) \*par, [fRect](#) rel)  
*attach a frame to a parent by relative position*
- void [plug](#) ([Frame](#) \*par)  
*attach a frame to a parent by old relative position*
- void [unplug](#) ()  
*detach a frame from its parent*
- void [moveTo](#) ([fPoint](#) rel)
- void [moveTo](#) (int x, int y)
- void [moveCenterTo](#) ([fPoint](#) rel)
- void [moveCenterTo](#) (int x, int y)
- void [moveBy](#) ([fPoint](#) rel)
- void [moveBy](#) (int, int)
- void [resize](#) (int w, int h)
- const Rectangle & [getFrame](#) () const
- const [fRect](#) & [getRelative](#) () const
- [Frame](#) \* [getParent](#) ()
- void [setRelative](#) ([fRect](#) rel)
- const [fPoint](#) & [getCenter](#) () const
- const float & [getX](#) () const
- const float & [getY](#) () const
- const float & [getW](#) () const
- const float & [getH](#) () const
- [operator Rectangle](#) () const
- [operator fRect](#) () const
- [operator iRect](#) () const

## Protected Member Functions

- void `fitFrame` ()
- void `updateFrame` (bool recursive=false) override
- bool `isroot` () const  
*return true if this frame is root*
- void `addSubframe` (Frame \*subframe)  
*Add a subframe to this frame.*
- void `removeSubframe` (Frame \*subframe)  
*Remove a subframe from this frame.*
- void `beginUpdate` ()
- void `endUpdate` ()

### 7.23.1 Detailed Description

images displayed on screen

Definition at line 16 of file `visual.hpp`.

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 Visual() [1/4]

```
Visual::Visual (
    Texture2D * txtr,
    Frame * frame,
    Rectangle rect )
```

Definition at line 4 of file `constructor.cpp`.

```
00004                                     : Frame(frame, rect)
00005 {
00006     m_texture = std::shared_ptr<Texture2D>(txtr, [](Texture2D* texture){
00007         UnloadTexture(*texture);
00008         texture = nullptr;
00009     });
00010     resizeable = true;
00011     fitFrame();
00012 }
```

#### 7.23.2.2 Visual() [2/4]

```
Visual::Visual (
    Visual * visual )
```

Definition at line 14 of file `constructor.cpp`.

```
00014                                     : Frame(visual)
00015 {
00016     resizeable = false;
00017     m_texture = visual->m_texture;
00018     fitFrame();
00019 }
```

### 7.23.2.3 Visual() [3/4]

```
Visual::Visual (
    Visual * visual,
    Rectangle rect )
```

Definition at line 21 of file [constructor.cpp](#).

```
00021                                     : Frame(visual, rect)
00022 {
00023     resizeable = false;
00024     m_texture = visual->m_texture;
00025     fitFrame();
00026 }
```

### 7.23.2.4 Visual() [4/4]

```
Visual::Visual (
    Visual * visual,
    Frame * frame,
    Rectangle rect )
```

Definition at line 28 of file [constructor.cpp](#).

```
00028                                     : Frame(frame, rect)
00029 {
00030     resizeable = false;
00031     m_texture = visual->m_texture;
00032     fitFrame();
00033 }
```

### 7.23.2.5 ~Visual()

```
Visual::~Visual ( )
```

Definition at line 12 of file [destructor.cpp](#).

```
00013 {
00014     m_texture.reset();
00015 }
```

## 7.23.3 Member Function Documentation

### 7.23.3.1 addSubframe()

```
void Frame::addSubframe (
    Frame * subframe ) [protected], [inherited]
```

Add a subframe to this frame.

When unplug a subframe, parent frame will call this function, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to add
-----------------	-----------------

Definition at line 70 of file [family.cpp](#).

```
00071 {
```

```

00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }

```

### 7.23.3.2 beginUpdate()

```
void Frame::beginUpdate ( ) [protected], [inherited]
```

Definition at line 113 of file [family.cpp](#).

```

00114 {
00115     mtx.lock();
00116 }

```

### 7.23.3.3 draw()

```
void Visual::draw ( )
```

Definition at line 4 of file [arithmetic.cpp](#).

```

00005 {
00006     if(m_texture == nullptr) return ;
00007     Rectangle rec = getFrame();
00008     // draw texture
00009     DrawTexture(*m_texture, rec.x, rec.y, WHITE);
00010 }

```

### 7.23.3.4 endUpdate()

```
void Frame::endUpdate ( ) [protected], [inherited]
```

Definition at line 118 of file [family.cpp](#).

```

00119 {
00120     mtx.unlock();
00121 }

```

### 7.23.3.5 fitFrame()

```
void Visual::fitFrame ( ) [protected]
```

Definition at line 12 of file [arithmetic.cpp](#).

```

00013 {
00014     if(m_texture == nullptr) return ;
00015     if(!resizable) return ;
00016     const Rectangle &rec = Frame::getFrame();
00017
00018     Image img = LoadImageFromTexture(*m_texture);
00019     UnloadTexture(*m_texture.get());
00020
00021     ImageResize(&img, rec.width, rec.height);
00022     *m_texture.get() = LoadTextureFromImage(img);
00023     UnloadImage(img);
00024 }

```

### 7.23.3.6 getCenter()

```
const fPoint & Frame::getCenter ( ) const [inherited]
```

Definition at line 131 of file [arithmetic.cpp](#).

```

00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139
00140     return resu;
00141 }

```

### 7.23.3.7 getFrame()

const Rectangle & Frame::getFrame ( ) const [inherited]

Definition at line 105 of file [arithmetic.cpp](#).

```
00106 {  
00107     std::lock_guard<std::mutex> lock(mtx);  
00108     return frame;  
00109 }
```

### 7.23.3.8 getH()

const float & Frame::getH ( ) const [inherited]

Definition at line 161 of file [arithmetic.cpp](#).

```
00162 {  
00163     std::lock_guard<std::mutex> lock(mtx);  
00164     return frame.height;  
00165 }
```

### 7.23.3.9 getParent()

Frame \* Frame::getParent ( ) [inherited]

Definition at line 117 of file [arithmetic.cpp](#).

```
00118 {  
00119     std::lock_guard<std::mutex> lock(mtx);  
00120     return parent;  
00121 }
```

### 7.23.3.10 getRelative()

const fRect & Frame::getRelative ( ) const [inherited]

Definition at line 111 of file [arithmetic.cpp](#).

```
00112 {  
00113     std::lock_guard<std::mutex> lock(mtx);  
00114     return relative;  
00115 }
```

### 7.23.3.11 getW()

const float & Frame::getW ( ) const [inherited]

Definition at line 155 of file [arithmetic.cpp](#).

```
00156 {  
00157     std::lock_guard<std::mutex> lock(mtx);  
00158     return frame.width;  
00159 }
```

### 7.23.3.12 getX()

const float & Frame::getX ( ) const [inherited]

Definition at line 143 of file [arithmetic.cpp](#).

```
00144 {  
00145     std::lock_guard<std::mutex> lock(mtx);  
00146     return frame.x;  
00147 }
```

**7.23.3.13 getY()**

```
const float & Frame::getY ( ) const [inherited]
```

Definition at line 149 of file [arithmetic.cpp](#).

```
00150 {
00151     std::lock_guard<std::mutex> lock(mtx);
00152     return frame.y;
00153 }
```

**7.23.3.14 isroot()**

```
bool Frame::isroot ( ) const [protected], [inherited]
```

return true if this frame is root

Definition at line 107 of file [family.cpp](#).

```
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
```

**7.23.3.15 moveBy() [1/2]**

```
void Frame::moveBy (
    fPoint rel ) [inherited]
```

Definition at line 65 of file [arithmetic.cpp](#).

```
00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }
```

**7.23.3.16 moveBy() [2/2]**

```
void Frame::moveBy (
    int x,
    int y ) [inherited]
```

Definition at line 75 of file [arithmetic.cpp](#).

```
00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }
```

**7.23.3.17 moveCenterTo() [1/2]**

```
void Frame::moveCenterTo (
    fPoint rel ) [inherited]
```

Definition at line 43 of file [arithmetic.cpp](#).

```
00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }
```

**7.23.3.18 moveCenterTo() [2/2]**

```
void Frame::moveCenterTo (
    int x,
    int y ) [inherited]
```

Definition at line 54 of file [arithmetic.cpp](#).

```
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
```

**7.23.3.19 moveTo() [1/2]**

```
void Frame::moveTo (
    fPoint rel ) [inherited]
```

Definition at line 24 of file [arithmetic.cpp](#).

```
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
```

**7.23.3.20 moveTo() [2/2]**

```
void Frame::moveTo (
    int x,
    int y ) [inherited]
```

Definition at line 33 of file [arithmetic.cpp](#).

```
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
```

**7.23.3.21 operator fRect()**

```
Frame::operator fRect ( ) const [inherited]
```

Definition at line 173 of file [arithmetic.cpp](#).

```
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
```

**7.23.3.22 operator iRect()**

```
Frame::operator iRect ( ) const [inherited]
```

Definition at line 179 of file [arithmetic.cpp](#).

```
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
```

**7.23.3.23 operator Rectangle()**

```
Frame::operator Rectangle ( ) const [inherited]
```

Definition at line 167 of file [arithmetic.cpp](#).

```
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
```

**7.23.3.24 plug() [1/2]**

```
void Frame::plug (
    Frame * par ) [inherited]
```

attach a frame to a parent by old relative position

**Parameters**

<i>par</i>	parent frame
------------	--------------

Definition at line 34 of file [family.cpp](#).

```
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
```

**7.23.3.25 plug() [2/2]**

```
void Frame::plug (
    Frame * par,
    fRect rel ) [inherited]
```

attach a frame to a parent by relative position

**Parameters**

<i>par</i>	parent frame
<i>rel</i>	relative position and size in percentage (0.0f to 1.0f)



Definition at line 12 of file [family.cpp](#).

```
00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }
```

### 7.23.3.26 removeSubframe()

```
void Frame::removeSubframe (
    Frame * subframe ) [protected], [inherited]
```

Remove a subframe from this frame.

When destroy a subframe that have parent frame, this function is called, so you shouldn't call it

#### Parameters

<i>subframe</i>	subframe to remove
-----------------	--------------------

Definition at line 85 of file [family.cpp](#).

```
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
```

### 7.23.3.27 resize() [1/2]

```
void Visual::resize (
    fPoint rel )
```

Definition at line 26 of file [arithmetic.cpp](#).

```
00027 {
00028     Frame::resize(rel);
00029     updateFrame(true);
00030 }
```

### 7.23.3.28 resize() [2/2]

```
void Frame::resize (
    int w,
    int h ) [inherited]
```

Definition at line 95 of file [arithmetic.cpp](#).

```
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
```

### 7.23.3.29 setRelative()

```
void Frame::setRelative (
    fRect rel ) [inherited]
```

Definition at line 123 of file [arithmetic.cpp](#).

```
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
```

### 7.23.3.30 unplug()

```
void Frame::unplug ( ) [inherited]
```

detach a frame from its parent

Definition at line 53 of file [family.cpp](#).

```
00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
```

### 7.23.3.31 updateFrame()

```
void Visual::updateFrame (
    bool recursive = false ) [override], [protected], [virtual]
```

Reimplemented from [Frame](#).

Definition at line 32 of file [arithmetic.cpp](#).

```
00033 {
00034     if(m_texture == nullptr) return ;
00035     float prx = getFrame().width;
00036     float pry = getFrame().height;
00037     Frame::updateFrame(recursive);
00038
00039     float rx = getFrame().width - prx;
00040     float ry = getFrame().height - pry;
00041     if(rx < 1e-3 && ry < 1e-3) return ;
00042     fitFrame();
00043 }
```

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

- [src/visual/include/visual.hpp](#)
- [src/visual/src/arithmetic.cpp](#)
- [src/visual/src/constructor.cpp](#)
- [src/visual/src/destructor.cpp](#)

## 7.24 Window Class Reference

```
#include <window.hpp>
```

### Public Member Functions

- [Window](#) ()
- [Window](#) (std::string path)
- [~Window](#) ()
- bool [isRun](#) ()
- bool [isClose](#) ()
- void [run](#) ()

### Protected Member Functions

- void [draw](#) ()
- void [systemEvent](#) ()
- void [getUserEvent](#) ()
- void [getRuntimeEvent](#) ()
- void [sound\\_effect](#) ()
- void [immediateActing](#) ()
- void [userActing](#) ()
- void [requestActing](#) ()
- void [systemActing](#) ()
- void [initRaylib](#) (YAML::Node node)
- void [loadInterface](#) (YAML::Node node)
- void [loadGame](#) (YAML::Node node)

### Friends

- class [CloseAction](#)
- class [resizeAction](#)

### 7.24.1 Detailed Description

Definition at line 21 of file [window.hpp](#).

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 Window() [1/2]

```
Window::Window ( )
```

Definition at line 7 of file [constructor.cpp](#).

```
00008 {
00009     Wcontent.width = 1200;
00010     Wcontent.height = 668;
00011     Wcontent.title = "Crossy Road clone";
00012     SetConfigFlags(FLAG_WINDOW_RESIZABLE | FLAG_VSYNC_HINT);
00013     InitWindow(Wcontent.width, Wcontent.height, Wcontent.title.c_str());
00014     SetTargetFPS(60);
00015
00016     UI.setRootFrame(new Frame{{0, 0, Wcontent.width, Wcontent.height}});
00017 }
```

### 7.24.2.2 Window() [2/2]

```
Window::Window (
    std::string path )
```

Definition at line 19 of file [constructor.cpp](#).

```
00020 {
00021     path = PATB::WINDOW_ + path;
00022     YAML::Node config = YAML_FILE::readFile(path);
00023     initRaylib(config);
00024
00025     loadInterface(config["interface-list"]);
00026     loadGame(config["game"]);
00027     if(config["choose-interface"])
00028         UI.push(config["choose-interface"].as<std::string>());
00029
00030     if(config["input-delay"])
00031     {
00032         double delay = config["input-delay"].as<int>() / 1000.0;
00033         Wcontent.input_delay = std::chrono::duration<double>(delay);
00034     }else
00035     {
00036         Wcontent.input_delay = std::chrono::duration<int>(50) / 1000.0;
00037     }
00038
00039     if(config["runtime-delay"])
00040     {
00041         double delay = config["runtime-delay"].as<int>() / 1000.0;
00042         Wcontent.runtime_delay = std::chrono::duration<double>(delay);
00043     }else
00044     {
00045         Wcontent.runtime_delay = std::chrono::duration<int>(40) / 1000.0;
00046     }
00047 }
```

### 7.24.2.3 ~Window()

```
Window::~~Window ( )
```

Definition at line 3 of file [destructor.cpp](#).

```
00004 {
00005     CloseWindow();
00006 }
```

## 7.24.3 Member Function Documentation

### 7.24.3.1 draw()

```
void Window::draw ( ) [protected]
```

Definition at line 34 of file [running.cpp](#).

```
00035 {
00036     {
00037         BeginDrawing();
00038         UI.draw();
00039         EndDrawing();
00040     }
00041
00042 }
```

### 7.24.3.2 getRuntimeEvent()

```
void Window::getRuntimeEvent ( ) [protected]
```

Definition at line 97 of file [running.cpp](#).

```
00098 {
00099     while(isRun())
00100     {
00101         if(!Wcontent.isRuntimeDelayOver())
00102         {
00103             std::this_thread::sleep_for(std::chrono::milliseconds(10));
00104             continue;
00105         }
00106         Action* action = UI.getRuntimeEvent();
00107         if(action != nullptr)
00108         {
00109             if(action->isPackage())
00110             {
00111                 for(auto act : action->unpack())
00112                 {
00113                     if(act->isRequest())
00114                         request_pool.push(act);
00115                     else
00116                         immediate_user_pool.push(act);
00117                 }
00118             }
00119             else if (!action->isRequest())
00120                 immediate_pool.push(action);
00121         }
00122         Wcontent.setRuntimeClock2Now();
00123     }
00124 }
00125 }
```

### 7.24.3.3 getUserEvent()

```
void Window::getUserEvent ( ) [protected]
```

Definition at line 66 of file [running.cpp](#).

```
00067 {
00068     while(isRun())
00069     {
00070         if(!Wcontent.isInputDelayOver())
00071         {
00072             std::this_thread::sleep_for(std::chrono::milliseconds(10));
00073             continue;
00074         }
00075         Action* action = UI.react();
00076         if(action != nullptr)
00077         {
00078             if(action->isPackage())
00079             {
00080                 for(auto act : action->unpack())
00081                 {
00082                     if(act->isRequest())
00083                         request_pool.push(act);
00084                     else
00085                         immediate_user_pool.push(act);
00086                 }
00087             }
00088             else if(!action->isRequest())
00089                 immediate_user_pool.push(action);
00090         }
00091         Wcontent.setInputClock2Now();
00092     }
00093 }
00094 }
00095 }
```

### 7.24.3.4 immediateActing()

```
void Window::immediateActing ( ) [protected]
```

Definition at line 50 of file [acting.cpp](#).

```
00051 {
00052     while(isRun())
00053     {
00054         Action* action = immediate_pool.pop();
00055         if(action == nullptr) continue;
00056         if(!isRun()) break;
00057         action->execute();
00058         delete action;
00059     }
00060 }
```

### 7.24.3.5 initRaylib()

```
void Window::initRaylib (
    YAML::Node node ) [protected]
```

Definition at line 50 of file [constructor.cpp](#).

```
00051 {
00052     Wcontent.width = config["width"].as<int>();
00053     Wcontent.height = config["height"].as<int>();
00054     Wcontent.title = config["title"].as<std::string>();
00055     // enable resizable window and vsync
00056
00057     SetConfigFlags(FLAG_WINDOW_RESIZABLE | FLAG_VSYNC_HINT);
00058     InitWindow(Wcontent.width, Wcontent.height, Wcontent.title.c_str());
00059     SetTargetFPS(60);
00060     Wcontent.setStatus(true);
00061     UI.setRootFrame(new Frame({0, 0, Wcontent.width, Wcontent.height}));
00062 }
```

### 7.24.3.6 isClose()

```
bool Window::isClose ( )
```

Definition at line 136 of file [running.cpp](#).

```
00137 {
00138     return !Wcontent.getStatus();
00139 }
```

### 7.24.3.7 isRun()

```
bool Window::isRun ( )
```

Definition at line 131 of file [running.cpp](#).

```
00132 {
00133     return Wcontent.getStatus();
00134 }
```

### 7.24.3.8 loadGame()

```
void Window::loadGame (
    YAML::Node node ) [protected]
```

Definition at line 86 of file [constructor.cpp](#).

```
00087 {
00088     if(!node) return ;
00089
00090     std::string path = node["file"].as<std::string>();
00091     float x = 0, y = 0, w = 1, h = 1;
00092
00093     if(node["x"]) x = node["x"].as<float>() / 100;
00094     if(node["y"]) y = node["y"].as<float>() / 100;
00095     if(node["w"]) w = node["w"].as<float>() / 100;
00096     if(node["h"]) h = node["h"].as<float>() / 100;
00097
00098
00099     Interface* inf = new Game(UI.getRootFrame(), {x, y, w, h});
00100     inf->linkContent(path);
00101     UI.load(inf);
00102 }
```

### 7.24.3.9 loadInterface()

```
void Window::loadInterface (
    YAML::Node node ) [protected]
```

Definition at line 64 of file [constructor.cpp](#).

```
00065 {
00066     if(!node) return ;
00067
00068     UI.setInterfacePool(new InterfacePool());
00069     for(auto i : node)
00070     {
00071         std::string path = i["file"].as<std::string>();
00072         float x = 0, y = 0, w = 1, h = 1;
00073
00074         if(i["x"]) x = i["x"].as<float>() / 100;
00075         if(i["y"]) y = i["y"].as<float>() / 100;
00076         if(i["w"]) w = i["w"].as<float>() / 100;
00077         if(i["h"]) h = i["h"].as<float>() / 100;
00078
00079         Interface* inf = new Interface(UI.getRootFrame(), {x, y, w, h});
00080         inf->linkContent(path);
00081         UI.load(inf);
00082     }
00083 }
00084 }
```

### 7.24.3.10 requestActing()

```
void Window::requestActing ( ) [protected]
```

Definition at line 85 of file [acting.cpp](#).

```
00086 {
00087     while(isRun())
00088     {
00089         Action* action = request_pool.pop();
00090         if(action == nullptr) continue;
00091         if(!isRun()) break;
00092         switch(action->isRequest())
00093         {
00094             case (REQUEST::ID::NONE):
00095                 break;
00096             case (REQUEST::ID::INVALID):
00097                 break;
00098             case (REQUEST::ID::CHANGE_INF):
00099                 {
00100                     std::string id = action->getArgs().getInterfaceName();
00101                     UI.push(id);
00102                     break;
00103                 }
00104             default:
00105                 break;
00106         };
00107         delete action;
00108     }
00109 }
00110 }
```

### 7.24.3.11 run()

```
void Window::run ( )
```

Definition at line 4 of file [running.cpp](#).

```
00004 {
00005     // last_christmas = now()
00006     Wcontent.setInputClock2Now();
00007     Wcontent.setRuntimeClock2Now();
00008
00009     //Wcontent.thread_pool.push_back(std::thread(&Window::draw, this));
00010     Wcontent.thread_pool.push_back(std::thread(&Window::getUserEvent, this));
00011     Wcontent.thread_pool.push_back(std::thread(&Window::getRuntimeEvent, this));
00012     //Wcontent.thread_pool.push_back(std::thread(&Window::sound_effect, this));
00013     Wcontent.thread_pool.push_back(std::thread(&Window::userActing, this));
```

```

00014     Wcontent.thread_pool.push_back(std::thread(&Window::userActing, this));
00015     Wcontent.thread_pool.push_back(std::thread(&Window::immediateActing, this));
00016     Wcontent.thread_pool.push_back(std::thread(&Window::immediateActing, this));
00017     Wcontent.thread_pool.push_back(std::thread(&Window::immediateActing, this));
00018     Wcontent.thread_pool.push_back(std::thread(&Window::requestActing, this));
00019
00020     while (isRun())
00021     {
00022         draw();
00023         systemEvent();
00024         systemActing();
00025         //getUserEvent();
00026         //getRuntimeEvent();
00027         //sound_effect();
00028         //userActing();
00029         //immediateActing();
00030         std::this_thread::sleep_for(std::chrono::milliseconds(5));
00031     }
00032 }

```

### 7.24.3.12 sound\_effect()

void Window::sound\_effect ( ) [protected]

Definition at line 126 of file [running.cpp](#).

```

00127 {
00128     // do nothing
00129 }

```

### 7.24.3.13 systemActing()

void Window::systemActing ( ) [protected]

Definition at line 74 of file [acting.cpp](#).

```

00075 {
00076     {
00077         Action* action = system_pool.pop();
00078         if(action == nullptr) return;
00079         if(!isRun()) return;
00080         action->execute();
00081         delete action;
00082     }
00083 }

```

### 7.24.3.14 systemEvent()

void Window::systemEvent ( ) [protected]

Definition at line 44 of file [running.cpp](#).

```

00045 {
00046     {
00047         // alt + F4 to exit
00048         if (IsKeyDown(KEY_LEFT_ALT) && IsKeyDown(KEY_F4))
00049         {
00050             system_pool.push(new CloseAction(this));
00051         }
00052         if (WindowShouldClose())
00053         {
00054             system_pool.push(new CloseAction(this));
00055         }
00056         if (IsWindowResized() && !IsWindowFullscreen())
00057         {
00058             int width = GetScreenWidth();
00059             int height = GetScreenHeight();
00060             system_pool.push(new ResizeAction(this, width, height));
00061         }
00062     }
00063 }
00064 }

```



### 7.24.3.15 userActing()

```
void Window::userActing ( ) [protected]
```

Definition at line 62 of file [acting.cpp](#).

```
00063 {  
00064     while(isRun())  
00065     {  
00066         Action* action = immediate_user_pool.pop();  
00067         if(action == nullptr) continue;  
00068         if(!isRun()) break;  
00069         action->execute();  
00070         delete action;  
00071     }  
00072 }
```

## 7.24.4 Friends And Related Symbol Documentation

### 7.24.4.1 CloseAction

```
friend class CloseAction [friend]
```

Definition at line 144 of file [window.hpp](#).

### 7.24.4.2 resizeAction

```
friend class resizeAction [friend]
```

Definition at line 145 of file [window.hpp](#).

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

- [src/window/include/window.hpp](#)
- [src/window/src/acting.cpp](#)
- [src/window/src/constructor.cpp](#)
- [src/window/src/destructor.cpp](#)
- [src/window/src/running.cpp](#)

# Chapter 8

## File Documentation

### 8.1 README.md File Reference

### 8.2 src/action/include/action.hpp File Reference

```
#include <vector>
#include <string>
```

#### Classes

- struct [ARGS](#)  
*stores request information*
- class [Action](#)  
*manages the way an action is executed*
- class [PacketAction](#)  
*organize selected actions into a package*

#### Variables

- [ARGS NONE\\_ARGS](#)

#### 8.2.1 Variable Documentation

##### 8.2.1.1 NONE\_ARGS

[ARGS NONE\\_ARGS](#) [extern]

Definition at line 4 of file [action.cpp](#).

## 8.3 action.hpp

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

```

00001 #ifndef ACTION_HPP
00002 #define ACTION_HPP
00003
00004 #include <vector>
00005 #include <string>
00006
00013 struct ARGS
00014 {
00015     std::vector<std::string> str;
00016     std::vector<int> num;
00017     std::vector<void*> addr;
00018     ARGS() = default;
00019     ~ARGS() = default;
00020
00021     std::string getInterfaceName();
00022 };
00023 extern ARGS NONE_ARGS;
00024
00031 class Action
00032 {
00033 public:
00034     Action();
00035     Action(Action*);
00036     virtual ~Action() = default;
00037
00038     virtual int isRequest();
00039     virtual bool isPackage();
00040     virtual void execute();
00041     virtual Action* clone();
00042     virtual std::vector<Action*> unpack();
00043     virtual ARGS& getArgs();
00044 };
00045
00052 class PacketAction : public Action
00053 {
00054 private:
00055     std::vector<Action*> actions;
00056 public:
00057     PacketAction();
00058     PacketAction(PacketAction*);
00059     ~PacketAction();
00060     bool isPackage() override;
00061     void addAction(Action*);
00062     void addAction(PacketAction*);
00063     std::vector<Action*> unpack() override;
00064     void execute() override;
00065     PacketAction* clone() override;
00066 };
00067
00068 #endif

```

## 8.4 src/action/include/request.hpp File Reference

```
#include <action.hpp>
```

### Classes

- class [Request](#)  
*sends information to a higher, relevant entity*
- class [changeInfRequest](#)
- class [loseRequest](#)  
*request sent when the player loses*

## 8.5 request.hpp

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

```
00001 #ifndef REQUEST_MY_HPP
00002 #define REQUEST_MY_HPP
00003
00004 #include <action.hpp>
00005
00014 class Request : public Action
00015 {
00016 protected:
00017     ARGS args;
00018 public:
00019     Request();
00020     Request(Request*);
00021     ~Request() = default;
00022
00023     int isRequest() override;
00024     virtual Action* clone() override;
00025 };
00026
00027 class changeInfRequest : public Request
00028 {
00029 public:
00030     changeInfRequest(std::string s);
00031     changeInfRequest(changeInfRequest*);
00032     ~changeInfRequest() = default;
00033     int isRequest() override;
00034     Action* clone() override;
00035     ARGS& getArgs() override;
00036 };
00037
00044 class loseRequest : public Request
00045 {
00046 public:
00047     loseRequest() = default;
00048     loseRequest(loseRequest*);
00049     ~loseRequest() = default;
00050     int isRequest() override;
00051     Action* clone() override;
00052 };
00053
00054 #endif
```

## 8.6 src/utls/include/const/request.hpp File Reference

### Namespaces

- namespace [REQUEST](#)

### Enumerations

- enum [REQUEST::ID](#) {  
[REQUEST::INVALID](#) , [REQUEST::NONE](#) , [REQUEST::CHANGE\\_INF](#) , [REQUEST::DELAY](#) ,  
[REQUEST::LOSE](#) }

## 8.7 request.hpp

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

```
00001 #ifndef REQUEST_HPP
00002 #define REQUEST_HPP
00003
00004 namespace REQUEST
00005 {
00006     enum ID
00007     {
00008         INVALID,
```

```
00009         NONE,  
00010         CHANGE_INF,  
00011         DELAY,  
00012         LOSE,  
00013     };  
00014 }  
00015  
00016 #endif  
00017
```

## 8.8 src/action/src/action.cpp File Reference

```
#include <action.hpp>  
#include <queue>
```

### Variables

- [ARGS NONE\\_ARGS](#)

### 8.8.1 Variable Documentation

#### 8.8.1.1 NONE\_ARGS

[ARGS](#) NONE\_ARGS

Definition at line 4 of file [action.cpp](#).

## 8.9 action.cpp

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

```
00001 #include <action.hpp>  
00002 #include <queue>  
00003  
00004 ARGS NONE_ARGS;  
00005  
00006 Action::Action()  
00007 {  
00008 }  
00009  
00010 Action::Action(Action* action)  
00011 {  
00012 }  
00013  
00014  
00015 int Action::isRequest()  
00016 {  
00017     return 0;  
00018 }  
00019  
00020 bool Action::isPackage()  
00021 {  
00022     return false;  
00023 }  
00024  
00025 void Action::execute()  
00026 {  
00027 }  
00028  
00029 Action* Action::clone()  
00030 {  
00031     return this;  
00032 }
```

```

00033
00034 std::vector<Action*> Action::unpack()
00035 {
00036     return std::vector<Action*> ({this});
00037 }
00038
00039 ARGS& Action::getArgs()
00040 {
00041     return NONE_ARGS;
00042 }
00043
00044 PacketAction::PacketAction() : Action()
00045 {
00046 }
00047
00048
00049 PacketAction::PacketAction(PacketAction* action) : Action(action)
00050 {
00051     for(Action* a : action->actions)
00052         actions.push_back(a->clone());
00053 }
00054
00055 PacketAction::~PacketAction()
00056 {
00057     for(Action* a : actions)
00058         delete a;
00059     actions.clear();
00060 }
00061
00062 bool PacketAction::isPackage()
00063 {
00064     return true;
00065 }
00066
00067 void PacketAction::addAction(Action* action)
00068 {
00069     actions.push_back(action);
00070 }
00071
00072 void PacketAction::addAction(PacketAction* action)
00073 {
00074     for(auto i : action->actions)
00075         actions.push_back(i);
00076     action->actions.clear();
00077 }
00078
00079 std::vector<Action*> PacketAction::unpack()
00080 {
00081     std::vector<Action*> unpacked;
00082     std::queue<PacketAction*> q;
00083
00084     q.push(this);
00085
00086     while(!q.empty())
00087     {
00088         PacketAction* p = q.front();
00089         q.pop();
00090
00091         for(Action* a : p->actions)
00092         {
00093             if(a->isPackage())
00094             {
00095                 q.push((PacketAction*)a);
00096             }
00097             else
00098             {
00099                 unpacked.push_back(a);
00100             }
00101         }
00102         p->actions.clear();
00103     }
00104     return unpacked;
00105 }
00106
00107 void PacketAction::execute()
00108 {
00109     for(Action* a : actions)
00110     {
00111         a->execute();
00112     }
00113 }
00114
00115
00116 PacketAction* PacketAction::clone()
00117 {
00118     return new PacketAction(this);
00119 }

```

## 8.10 src/chunk/src/action.cpp File Reference

```
#include "action.hpp"
#include <chunk.hpp>
```

## 8.11 action.cpp

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

```
00001 #include "action.hpp"
00002 #include <chunk.hpp>
00003
00004 Container* Chunk::randomEntity()
00005 {
00006     int value = GetRandomValue(0, 100);
00007
00008     for(auto i : visitor)
00009     {
00010         value -= i->getProbability();
00011         if(value <= 0) return i;
00012     }
00013
00014     return visitor[GetRandomValue(0, visitor.size() - 1)];
00015 }
00016
00017 void Chunk::movingEntity()
00018 {
00019     for(auto i : Entity)
00020     {
00021         i->moveBy(velocity);
00022         i->nextImage();
00023     }
00024 }
00025
00026 Action* Chunk::getRuntimeEvent()
00027 {
00028     PacketAction* packet = nullptr;
00029     Action* action = Interface::getRuntimeEvent();
00030
00031     if(action != nullptr)
00032     {
00033         packet = new PacketAction();
00034         packet->addAction(action);
00035     }
00036
00037     if(std::chrono::system_clock::now() - moveClock >= moveTime)
00038     {
00039         Action* action = new moveEntityAction(this);
00040         if(packet == nullptr)
00041         {
00042             packet = new PacketAction();
00043         }
00044         packet->addAction(action);
00045         moveClock = std::chrono::system_clock::now();
00046     }
00047     return packet;
00048 }
```

## 8.12 src/game/src/action.cpp File Reference

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

## 8.13 action.cpp

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

```

00001 #include <game.hpp>
00002
00003 Action* Game::react()
00004 {
00005     return Interface::react();
00006 }
00007
00008 Action* Game::getRuntimeEvent()
00009 {
00010     // if now - mapSpeedClock < 10 millisecond, return nullptr
00011
00012     if(std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now() -
mapSpeedClock).count() < 20)
00013         return nullptr;
00014     Action* action;
00015     PacketAction* packet = nullptr;
00016     action = Interface::getRuntimeEvent();
00017
00018     if(action != nullptr)
00019     {
00020         packet = new PacketAction();
00021         packet->addAction(action);
00022     }
00023
00024     action = new moveChunksAction(this, mapDisplacement);
00025     if(packet == nullptr) packet = new PacketAction();
00026     packet->addAction(action);
00027
00028     for(auto i : chunks)
00029     {
00030         Action* act = i->getRuntimeEvent();
00031         if(act == nullptr)
00032             continue;
00033         if(packet == nullptr)
00034             packet = new PacketAction();
00035         packet->addAction(act);
00036     }
00037
00038     action = new moveObjectAction(main, mapDisplacement);
00039     if(packet == nullptr) packet = new PacketAction();
00040     packet->addAction(action);
00041
00042     mapSpeedClock = std::chrono::system_clock::now();
00043
00044     return packet;
00045 }

```

## 8.14 src/interface/src/action.cpp File Reference

```

#include "action.hpp"
#include <interface.hpp>

```

## 8.15 action.cpp

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

```

00001 #include "action.hpp"
00002 #include <interface.hpp>
00003
00004 Action* Interface::getRuntimeEvent()
00005 {
00006     PacketAction* packet = nullptr;
00007     Action* action = Container::getRuntimeEvent();
00008
00009     if(action != nullptr)
00010     {
00011         packet = new PacketAction();
00012         packet->addAction(action);
00013     }

```



```

00014
00015     for(auto i : nested)
00016     {
00017         action = i->getRuntimeEvent();
00018         if(action != nullptr)
00019         {
00020             if(packet == nullptr) packet = new PacketAction();
00021             packet->addAction(action);
00022         }
00023     }
00024
00025     for(auto i : containers)
00026     {
00027         action = i->getRuntimeEvent();
00028         if(action != nullptr)
00029         {
00030             if(packet == nullptr) packet = new PacketAction();
00031             packet->addAction(action);
00032         }
00033     }
00034
00035     return packet;
00036 }
00037
00038 Action* Interface::react()
00039 {
00040     if(!isVisible()) return nullptr;
00041     PacketAction* packet = nullptr;
00042
00043     Action* action = Container::react();
00044
00045     if(action != nullptr)
00046     {
00047         packet = new PacketAction();
00048         packet->addAction(action);
00049     }
00050
00051     for(auto i : keystrokes)
00052     {
00053         Action* action = i->react();
00054         if(action != nullptr)
00055         {
00056             if(packet == nullptr) packet = new PacketAction();
00057             packet->addAction(action);
00058         }
00059     }
00060
00061     for(auto i : containers)
00062     {
00063         Action* action = i->react();
00064         if(action != nullptr)
00065         {
00066             if(packet == nullptr) packet = new PacketAction();
00067             packet->addAction(action);
00068         }
00069     }
00070
00071     return packet;
00072 }

```

## 8.16 src/action/src/args.cpp File Reference

```
#include <action.hpp>
```

## 8.17 args.cpp

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

```

00001 #include <action.hpp>
00002
00003 std::string ARGS::getInterfaceName()
00004 {
00005     return str[0];
00006 }

```

## 8.18 src/action/src/request.cpp File Reference

```
#include <request.hpp>
#include <const/request.hpp>
```

## 8.19 request.cpp

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

```
00001 #include <request.hpp>
00002 #include <const/request.hpp>
00003
00004 Request::Request() : Action()
00005 {
00006 }
00007
00008 Request::Request(Request* request) : Action(request)
00009 {
00010 }
00011
00012 int Request::isRequest()
00013 {
00014     return 1;
00015 }
00016
00017 Action* Request::clone()
00018 {
00019     return new Request(this);
00020 }
```

## 8.20 src/utlis/src/request.cpp File Reference

## 8.21 request.cpp

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

## 8.22 src/action/src/request/changeinf.cpp File Reference

```
#include <request.hpp>
#include <const/request.hpp>
```

## 8.23 changeinf.cpp

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

```
00001 #include <request.hpp>
00002 #include <const/request.hpp>
00003 changeInfRequest::changeInfRequest(std::string s)
00004 {
00005     args.str.push_back(s);
00006 }
00007
00008 changeInfRequest::changeInfRequest(changeInfRequest* other)
00009 {
00010     args = other->args;
00011 }
```

```

00012
00013 int changeInfRequest::isRequest()
00014 {
00015     return REQUEST::CHANGE_INF;
00016 }
00017
00018 Action* changeInfRequest::clone()
00019 {
00020     return new changeInfRequest(this);
00021 }
00022
00023 ARGS& changeInfRequest::getArgs()
00024 {
00025     return args;
00026 }

```

## 8.24 src/action/src/request/lose.cpp File Reference

```

#include <request.hpp>
#include <const/request.hpp>

```

## 8.25 lose.cpp

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

```

00001 #include <request.hpp>
00002 #include <const/request.hpp>
00003
00004 loseRequest::loseRequest(loseRequest* other)
00005 {
00006     args = other->args;
00007 }
00008
00009 int loseRequest::isRequest()
00010 {
00011     return REQUEST::LOSE;
00012 }
00013
00014 Action* loseRequest::clone()
00015 {
00016     return new loseRequest(this);
00017 }
00018

```

## 8.26 src/button/include/button.hpp File Reference

```

#include <raylib.h>
#include <frame.hpp>
#include <container.hpp>

```

### Classes

- class [ButtonImage](#)  
*manages the appearance and behavior of a button*

### Macros

- #define [TRANSPARENT](#) Color {127, 127, 127, 0}
- #define [rectangle](#) this->getFrame()

## 8.26.1 Macro Definition Documentation

### 8.26.1.1 rectangle

```
#define rectangle this->getFrame()
```

Definition at line 10 of file [button.hpp](#).

### 8.26.1.2 TRANSPARENT

```
#define TRANSPARENT Color {127, 127, 127, 0}
```

Definition at line 9 of file [button.hpp](#).

## 8.27 button.hpp

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

```
00001 #ifndef BUTTON_HPP
00002 #define BUTTON_HPP
00003
00004 #include <raylib.h>
00005
00006 #include <frame.hpp>
00007 #include <container.hpp>
00008
00009 #define TRANSPARENT Color {127, 127, 127, 0}
00010 #define rectangle this->getFrame()
00011
00012 class ButtonImage : public Container
00013 {
00014 private:
00015     static constexpr int DPI = 500;
00016     static constexpr float CORNER_RADIUS = 0.3;
00017
00018     std::vector <std::string> path;
00019     std::vector <std::string> pathPress;
00020
00021     int numpath;
00022     int tmpPath;
00023     int releaseID;
00024     int hoverID;
00025     int pressingID;
00026     int clickedID;
00027
00028     // Rectangle rectangle;
00029     Color color;
00030
00031     bool isHover = false;
00032     bool pressing = false, clicked = false;
00033
00034     std::vector <Action*> actions;
00035
00036 protected:
00037     void loadEvent(YAML::Node node);
00038
00039 public:
00040     ButtonImage(Frame* parent, Rectangle relative);
00041     ~ButtonImage();
00042     void draw();
00043     PacketAction* react();
00044
00045     void changeIndex(int newIndex);
00046     void changePosition(Rectangle change);
00047     [[nodiscard]] bool isClicked() const;
00048     [[nodiscard]] bool isPressing() const;
00049     int getClicked();
00050
00051     std::string linkContent(std::string);
00052     std::string linkContentAbsolute(std::string);
00053 };
00054
00055 #endif
```

## 8.28 src/button/src/arithmic.cpp File Reference

```
#include <button.hpp>
#include <request.hpp>
```

## 8.29 arithmic.cpp

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

```
00001 #include <button.hpp>
00002 #include <request.hpp>
00003
00004
00005 // Button for image
00006
00007 void ButtonImage::draw() {
00008     if(!isVisible()) return;
00009     this->Container::draw();
00010 }
00011
00012 PacketAction* ButtonImage::react() {
00013
00014     if (CheckCollisionPointRec(GetMousePosition(), rectangle)) {
00015         this->isHover = 1;
00016         if (IsMouseButtonDown(MOUSE_LEFT_BUTTON)) { // click -> pressing
00017             this->clicked = true;
00018             if(this->pressingID == -1)
00019                 return nullptr;
00020             PacketAction* packet = new PacketAction();
00021             packet->addAction(actions[pressingID]->clone());
00022             return packet;
00023         }
00024         else if(this->clicked) { // release -> click
00025
00026             this->clicked = false;
00027             if(this->clickedID == -1)
00028                 return nullptr;
00029             PacketAction* packet = new PacketAction();
00030             packet->addAction(actions[clickedID]->clone());
00031             packet->addAction(new changeInfRequest("test"));
00032             return packet;
00033         }
00034         if(this->hoverID == -1)
00035             return nullptr;
00036         PacketAction* packet = new PacketAction();
00037         packet->addAction(actions[hoverID]->clone());
00038         return packet;
00039     }
00040     if (this->isHover == 1)
00041     {
00042         this->isHover = 0;
00043         if(this->releaseID == -1)
00044             return nullptr;
00045         PacketAction* packet = new PacketAction();
00046         packet->addAction(actions[releaseID]->clone());
00047         return packet;
00048     }
00049     return nullptr;
00050 }
00051
00052
00053
00054 void ButtonImage::changeIndex(int newIndex)
00055 {
00056     tmpPath = newIndex;
00057 }
00058
00059 int ButtonImage::getClicked()
00060 {
00061     return tmpPath;
00062 }
00063
00064 bool ButtonImage::isClicked() const {
00065     return this->clicked;
00066 }
```

## 8.30 src/chunk/src/arithmic.cpp File Reference

```
#include <chunk.hpp>
```

### 8.31 arithmic.cpp

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

```
00001 #include <chunk.hpp>
00002
00003 void Chunk::drawEntity()
00004 {
00005     for(auto i : Entity)
00006         i->draw();
00007 }
00008
00009 void Chunk::draw()
00010 {
00011
00012     Container::draw();
00013     drawNested();
00014     drawContainers();
00015     drawEntity();
00016 }
```

## 8.32 src/container/src/arithmic.cpp File Reference

```
#include "action.hpp"
#include <container.hpp>
```

### 8.33 arithmic.cpp

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

```
00001 #include "action.hpp"
00002 #include <container.hpp>
00003
00004 void Container::draw()
00005 {
00006     if(sprites.empty()) return;
00007     if(!visible) return;
00008     sprites[focus[0]][focus[1]]->draw();
00009 }
00010
00011 void Container::show()
00012 {
00013     visible = true;
00014 }
00015
00016 void Container::hide()
00017 {
00018     visible = false;
00019 }
00020
00021 void Container::toggleVisibility()
00022 {
00023     visible = !visible;
00024 }
00025
00026 bool Container::isVisible()
00027 {
00028     return visible;
00029 }
00030
00031 int Container::getInstanceId()
```

```

00032 {
00033     return instance_id;
00034 }
00035
00036 Action* Container::react()
00037 {
00038     return nullptr;
00039 }
00040
00041 Action* Container::getRuntimeEvent()
00042 {
00043     return nullptr;
00044 }

```

## 8.34 src/frame/src/arithmic.cpp File Reference

```
#include <frame.hpp>
```

## 8.35 arithmic.cpp

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

```

00001 #include <frame.hpp>
00002
00003 void Frame::updateFrame(bool recursive)
00004 {
00005
00006     if(parent != nullptr)
00007     {
00008         std::lock_guard<std::mutex> lock(mtx);
00009         frame.x = parent->getX() + relative[0] * parent->getW();
00010         frame.y = parent->getY() + relative[1] * parent->getH();
00011         frame.width = relative[2] * parent->getW();
00012         frame.height = relative[3] * parent->getH();
00013     }
00014
00015     if(recursive)
00016     {
00017         for(auto& subframe : subframes)
00018         {
00019             subframe->updateFrame(true);
00020         }
00021     }
00022 }
00023
00024 void Frame::moveTo(fPoint rel)
00025 {
00026     if(isroot()) return ;
00027     mtx.lock();
00028     relative[0] = rel[0];
00029     relative[1] = rel[1];
00030     mtx.unlock();
00031     updateFrame(true);
00032 }
00033 void Frame::moveTo(int x, int y)
00034 {
00035     if(parent != nullptr) return ;
00036     mtx.lock();
00037     frame.x = x;
00038     frame.y = y;
00039     mtx.unlock();
00040     updateFrame(true);
00041 }
00042
00043 void Frame::moveCenterTo(fPoint rel)
00044 {
00045     if(isroot()) return ;
00046     mtx.lock();
00047     fPoint center = getCenter();
00048     relative[0] += rel[0] - center[0];
00049     relative[1] += rel[1] - center[1];
00050     mtx.unlock();
00051     updateFrame(true);
00052 }
00053

```

```

00054 void Frame::moveCenterTo(int x, int y)
00055 {
00056     if(parent != nullptr) return ;
00057     mtx.lock();
00058     fPoint center = getCenter();
00059     frame.x += x - center[0];
00060     frame.y += y - center[1];
00061     mtx.unlock();
00062     updateFrame(true);
00063 }
00064
00065 void Frame::moveBy(fPoint rel)
00066 {
00067     if(isroot()) return ;
00068     mtx.lock();
00069     relative[0] += rel[0];
00070     relative[1] += rel[1];
00071     mtx.unlock();
00072     updateFrame(true);
00073 }
00074
00075 void Frame::moveBy(int x, int y)
00076 {
00077     if(parent != nullptr) return ;
00078     mtx.lock();
00079     frame.x += x;
00080     frame.y += y;
00081     mtx.unlock();
00082     updateFrame(true);
00083 }
00084
00085 void Frame::resize(fPoint rel)
00086 {
00087     if(isroot()) return ;
00088     mtx.lock();
00089     relative[2] = rel[0];
00090     relative[3] = rel[1];
00091     mtx.unlock();
00092     updateFrame(true);
00093 }
00094
00095 void Frame::resize(int w, int h)
00096 {
00097     if(parent != nullptr) return ;
00098     mtx.lock();
00099     frame.width = w;
00100     frame.height = h;
00101     mtx.unlock();
00102     updateFrame(true);
00103 }
00104
00105 const Rectangle& Frame::getFrame() const
00106 {
00107     std::lock_guard<std::mutex> lock(mtx);
00108     return frame;
00109 }
00110
00111 const fRect& Frame::getRelative() const
00112 {
00113     std::lock_guard<std::mutex> lock(mtx);
00114     return relative;
00115 }
00116
00117 Frame* Frame::getParent()
00118 {
00119     std::lock_guard<std::mutex> lock(mtx);
00120     return parent;
00121 }
00122
00123 void Frame::setRelative(fRect rel)
00124 {
00125     mtx.lock();
00126     relative = rel;
00127     mtx.unlock();
00128     updateFrame(true);
00129 }
00130
00131 const fPoint& Frame::getCenter() const
00132 {
00133     std::lock_guard<std::mutex> lock(mtx);
00134     static fPoint resu;
00135     if(isroot())
00136         resu = {frame.x + frame.width / 2, frame.y + frame.height / 2};
00137     else
00138         resu = {relative[0] + relative[2] / 2, relative[1] + relative[3] / 2};
00139     return resu;
00140 }

```



```

00141 }
00142
00143 const float& Frame::getX() const
00144 {
00145     std::lock_guard<std::mutex> lock(mtx);
00146     return frame.x;
00147 }
00148
00149 const float& Frame::getY() const
00150 {
00151     std::lock_guard<std::mutex> lock(mtx);
00152     return frame.y;
00153 }
00154
00155 const float& Frame::getWidth() const
00156 {
00157     std::lock_guard<std::mutex> lock(mtx);
00158     return frame.width;
00159 }
00160
00161 const float& Frame::getHeight() const
00162 {
00163     std::lock_guard<std::mutex> lock(mtx);
00164     return frame.height;
00165 }
00166
00167 Frame::operator Rectangle() const
00168 {
00169     std::lock_guard<std::mutex> lock(mtx);
00170     return frame;
00171 }
00172
00173 Frame::operator fRect() const
00174 {
00175     std::lock_guard<std::mutex> lock(mtx);
00176     return relative;
00177 }
00178
00179 Frame::operator iRect() const
00180 {
00181     std::lock_guard<std::mutex> lock(mtx);
00182     return {(int) frame.x, (int) frame.y, (int) frame.width, (int) frame.height};
00183 }
00184
00185

```

## 8.36 src/game/src/arithmetic.cpp File Reference

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

## 8.37 arithmetic.cpp

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

```

00001 #include <game.hpp>
00002
00003 void Game::draw()
00004 {
00005
00006     drawNested();
00007
00008     for(auto i = chunks.begin(); i != chunks.end(); ++i)
00009     {
00010         (*i)->draw();
00011     }
00012
00013     drawContainers();
00014 }

```

## 8.38 src/interface/src/arithmetic.cpp File Reference

```
#include <interface.hpp>
```

## 8.39 arithmetic.cpp

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

```
00001 #include <interface.hpp>
00002
00003 void Interface::drawNested()
00004 {
00005     for(auto& child : nested)
00006     {
00007         child->draw();
00008     }
00009 }
00010
00011 void Interface::drawContainers()
00012 {
00013     for(auto& child : containers)
00014     {
00015         child->draw();
00016     }
00017 }
00018
00019 void Interface::draw()
00020 {
00021     Container::draw();
00022     drawNested();
00023     drawContainers();
00024 }
00025
00026
00027
00028 }
```

## 8.40 src/object/src/arithmetic.cpp File Reference

```
#include "container.hpp"
#include <object.hpp>
```

## 8.41 arithmetic.cpp

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

```
00001 #include "container.hpp"
00002 #include <object.hpp>
00003 Action* Object::react()
00004 {
00005     if(std::chrono::steady_clock::now() < waitUntil)
00006         return nullptr;
00007     for(int i = 0; i < strokes.size(); i++)
00008     {
00009         Action* a = strokes[i].stroke->react();
00010         if(a == nullptr) continue;
00011         else strokes[i].stroke->nextAction();
00012         return a;
00013     }
00014 }
00015
00016 return nullptr;
00017 }
00018
00019 void Object::draw()
00020 {
00021     Container::draw();
00022     return ;
00023 }
```

## 8.42 src/visual/src/arithmetic.cpp File Reference

```
#include <visual.hpp>
```

## 8.43 arthmetic.cpp

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

```
00001 #include <visual.hpp>
00002
00003
00004 void Visual::draw()
00005 {
00006     if(m_texture == nullptr) return ;
00007     Rectangle rec = getFrame();
00008     // draw texture
00009     DrawTexture(*m_texture, rec.x, rec.y, WHITE);
00010 }
00011
00012 void Visual::fitFrame()
00013 {
00014     if(m_texture == nullptr) return ;
00015     if(!resizable) return ;
00016     const Rectangle &rec = Frame::getFrame();
00017
00018     Image img = LoadImageFromTexture(*m_texture);
00019     UnloadTexture(*m_texture.get());
00020
00021     ImageResize(&img, rec.width, rec.height);
00022     *m_texture.get() = LoadTextureFromImage(img);
00023     UnloadImage(img);
00024 }
00025
00026 void Visual::resize(fPoint rel)
00027 {
00028     Frame::resize(rel);
00029     updateFrame(true);
00030 }
00031
00032 void Visual::updateFrame(bool recursive)
00033 {
00034     if(m_texture == nullptr) return ;
00035     float prx = getFrame().width;
00036     float pry = getFrame().height;
00037     Frame::updateFrame(recursive);
00038
00039     float rx = getFrame().width - prx;
00040     float ry = getFrame().height - pry;
00041     if(rx < 1e-3 && ry < 1e-3) return ;
00042     fitFrame();
00043 }
```

## 8.44 src/button/src/constructor.cpp File Reference

```
#include <button.hpp>
```

## 8.45 constructor.cpp

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

```
00001 #include <button.hpp>
00002
00003
00004 // Button for Image
00005 ButtonImage::ButtonImage(Frame* parrent, Rectangle rel) : Container(parrent, rel)
00006 {
00007     // set default
00008     this->chooseImage(0, this->tmpPath);
00009     this->color = WHITE;
00010     this->pressing = false;
00011     this->isHover = false;
00012     this->clicked = false;
00013
00014     this->releaseID = -1;
00015     this->hoverID = -1;
00016     this->pressingID = -1;
00017     this->clickedID = -1;
```

```

00018 }
00019
00020 std::string ButtonImage::linkContent(std::string path)
00021 {
00022     return linkContentAbsolute(PATB::BUTTON_ + path);
00023 }
00024
00025 std::string ButtonImage::linkContentAbsolute(std::string path)
00026 {
00027     YAML::Node node = YAML_FILE::readFile(path);
00028     if(!loadName(node)) return "";
00029
00030     if(node["textures"])
00031     {
00032         loadSprites(node["textures"]);
00033         chooseImage(0, 0);
00034     }
00035     if(node["events"])
00036     {
00037         loadEvent(node["events"]);
00038     }
00039
00040     return getName();
00041 }
00042
00043 void ButtonImage::loadEvent(YAML::Node node)
00044 {
00045     if(node["hover"])
00046     {
00047
00048         for(auto sprite : node["hover"]["sprite"])
00049         {
00050             iPoint p;
00051             int delay = 0;
00052             p[0] = sprite[0].as<int>();
00053             p[1] = sprite[1].as<int>();
00054             if(p.size() >= 3)
00055                 delay = sprite[2].as<int>();
00056             actions.push_back(new changeImageAction(this, p));
00057         }
00058         this->hoverID = actions.size() - 1;
00059     }
00060
00061     if(node["release"])
00062     {
00063         for(auto sprite : node["release"]["sprite"])
00064         {
00065             iPoint p;
00066             int delay = 0;
00067             p[0] = sprite[0].as<int>();
00068             p[1] = sprite[1].as<int>();
00069             if(p.size() >= 3)
00070                 delay = sprite[2].as<int>();
00071             actions.push_back(new changeImageAction(this, p));
00072         }
00073         this->releaseID = actions.size() - 1;
00074     }
00075
00076     if(node["clicked"])
00077     {
00078         for(auto sprite : node["clicked"]["sprite"])
00079         {
00080             iPoint p;
00081             int delay = 0;
00082             p[0] = sprite[0].as<int>();
00083             p[1] = sprite[1].as<int>();
00084             if(p.size() >= 3)
00085                 delay = sprite[2].as<int>();
00086             actions.push_back(new changeImageAction(this, p));
00087         }
00088         this->clickedID = actions.size() - 1;
00089     }
00090
00091     if(node["pressing"])
00092     {
00093         for(auto sprite : node["pressing"]["sprite"])
00094         {
00095             iPoint p;
00096             int delay = 0;
00097             p[0] = sprite[0].as<int>();
00098             p[1] = sprite[1].as<int>();
00099             if(p.size() >= 3)
00100                 delay = sprite[2].as<int>();
00101             actions.push_back(new changeImageAction(this, p));
00102         }
00103         this->pressingID = actions.size() - 1;
00104     }

```

```
00105     }
00106 }
```

## 8.46 src/chunk/src/constructor.cpp File Reference

```
#include "const/path/atb.hpp"
#include <chunk.hpp>
```

## 8.47 constructor.cpp

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

```
00001 #include "const/path/atb.hpp"
00002 #include <chunk.hpp>
00003
00004 Chunk::Chunk(Frame* frame, Rectangle rect) : Interface(frame, rect)
00005 {
00006
00007 }
00008
00009 Chunk::Chunk(Chunk* other) : Interface(other)
00010 {
00011
00012     for(auto i : other->visiter)
00013     {
00014         Rectangle rel;
00015         rel.x = 1;
00016         rel.y = -0.375;
00017         rel.width = i->getRelative()[2];
00018         rel.height = i->getRelative()[3];
00019         visiter.push_back(new Container(i, this, rel));
00020     }
00021     velocity = other->velocity;
00022     generateEntity();
00023 }
00024
00025 Chunk::Chunk(Chunk* other, Rectangle rect) : Interface(other, rect)
00026 {
00027     for(auto i : other->visiter)
00028     {
00029         Rectangle rel;
00030         rel.x = 1;
00031         rel.y = -0.375;
00032         rel.width = i->getRelative()[2];
00033         rel.height = i->getRelative()[3];
00034         visiter.push_back(new Container(i, this, rel));
00035     }
00036     velocity = other->velocity;
00037     generateEntity();
00038 }
00039
00040 Chunk::Chunk(Chunk* other, Frame* frame, Rectangle rect) : Interface(other, frame, rect)
00041 {
00042     for(auto i : other->visiter)
00043     {
00044         Rectangle rel;
00045         rel.x = 1;
00046         rel.y = -0.375;
00047         rel.width = i->getRelative()[2];
00048         rel.height = i->getRelative()[3];
00049         visiter.push_back(new Container(i, this, rel));
00050     }
00051     velocity = other->velocity;
00052     generateEntity();
00053 }
00054
00055 void Chunk::generateEntity()
00056 {
00057
00058     if(visiter.empty()) return;
00059     float x = GetRandomValue(-40, 10);
00060
00061     while(x < 0.9)
00062     {
```

```

00063         Container* c = randomEntity();
00064         Rectangle rel;
00065         rel.x = x;
00066         rel.y = -0.375;
00067         rel.width = c->getRelative()[2];
00068         rel.height = c->getRelative()[3];
00069         Container* cont = new Container(c, this, rel);
00070         Entity.push_back(cont);
00071         x += GetRandomValue(20, 60) / 100.0;
00072     }
00073 }
00074
00075
00076 std::string Chunk::linkContent(std::string path)
00077 {
00078     return linkContentAbsolute(PATB::CHUNK_ + path);
00079 }
00080
00081
00082 void Chunk::addVisitor(Container* obj)
00083 {
00084     Rectangle rel;
00085     rel.x = obj->getRelative()[0];
00086     rel.y = obj->getRelative()[1];
00087     rel.width = obj->getRelative()[2];
00088     rel.height = obj->getRelative()[3];
00089
00090     Container* c = new Container(obj, this, rel);
00091     visitor.push_back(c);
00092 }
00093
00094 void Chunk::addVisitor(Container* obj, int prob)
00095 {
00096     Rectangle rel;
00097     rel.x = obj->getRelative()[0];
00098     rel.y = obj->getRelative()[1];
00099     rel.width = obj->getRelative()[2];
00100     rel.height = obj->getRelative()[3];
00101
00102     Container* c = new Container(obj, this, rel);
00103     c->setProbability(prob);
00104     visitor.push_back(c);
00105 }
00106
00107 void Chunk::addVisitor(Container* obj, Rectangle rel)
00108 {
00109     Container* c = new Container(obj, this, rel);
00110     visitor.push_back(c);
00111 }
00112
00113 void Chunk::addVisitor(Container* obj, int prob, Rectangle rel)
00114 {
00115     Container* c = new Container(obj, this, rel);
00116     c->setProbability(prob);
00117     visitor.push_back(c);
00118 }
00119
00120 void Chunk::setVelocity(fPoint vel)
00121 {
00122     velocity = vel;
00123 }

```

## 8.48 src/container/src/constructor.cpp File Reference

```

#include "raylib.h"
#include <visual.hpp>
#include <container.hpp>
#include <const/path/atb.hpp>
#include <const/path/assets.hpp>
#include <file.hpp>

```

## 8.49 constructor.cpp

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

```

00001 #include "raylib.h"
00002 #include <visual.hpp>
00003 #include <container.hpp>
00004 #include <const/path/atb.hpp>
00005 #include <const/path/assets.hpp>
00006 #include <file.hpp>
00007
00008 int Container::id_count = 0;
00009
00010 Container::Container(Frame* parent, Rectangle rect) : Frame(parent, rect)
00011 {
00012     instance_id = id_count++;
00013     focus = {0, 0};
00014     visible = true;
00015 }
00016
00017 Container::Container(Container* other) : Frame(other)
00018 {
00019     instance_id = id_count++;
00020     focus = {0, 0};
00021     name = other->name;
00022     visible = true;
00023
00024     for(auto s : other->sprites)
00025     {
00026         sprites.emplace_back();
00027         Rectangle rect;
00028         rect.x = other->getRelative()[0];
00029         rect.y = other->getRelative()[1];
00030         rect.width = other->getRelative()[2];
00031         rect.height = other->getRelative()[3];
00032
00033         for(auto v : s)
00034         {
00035             sprites.back().push_back(new Visual(v, this, rect));
00036         }
00037     }
00038 }
00039
00040 Container::Container(Container* other, Rectangle rect) : Frame(other)
00041 {
00042     instance_id = id_count++;
00043     focus = {0, 0};
00044     name = other->name;
00045     setRelative({rect.x, rect.y, rect.width, rect.height});
00046     visible = true;
00047
00048     for(auto s : other->sprites)
00049     {
00050         sprites.emplace_back();
00051         Rectangle rect;
00052         rect.x = other->getRelative()[0];
00053         rect.y = other->getRelative()[1];
00054         rect.width = other->getRelative()[2];
00055         rect.height = other->getRelative()[3];
00056
00057         for(auto v : s)
00058         {
00059             sprites.back().push_back(new Visual(v, this, rect));
00060         }
00061     }
00062 }
00063
00064 Container::Container(Container* other, Frame* parent, Rectangle rect) : Frame(parent, rect)
00065 {
00066     instance_id = id_count++;
00067     focus = {0, 0};
00068     name = other->name;
00069     visible = true;
00070     for(auto s : other->sprites)
00071     {
00072         sprites.emplace_back();
00073         Rectangle rect;
00074         rect.x = other->getRelative()[0];
00075         rect.y = other->getRelative()[1];
00076         rect.width = other->getRelative()[2];
00077         rect.height = other->getRelative()[3];
00078
00079         for(auto v : s)
00080         {
00081             sprites.back().push_back(new Visual(v, this, rect));
00082         }
00083     }
00084 }
00085
00086 std::string Container::linkContent(std::string path)
00087 {

```

```

00088     focus = {0, 0};
00089     return linkContentAbsolute(PATB::CONTAINER_ + path);
00090 }
00091
00092 std::string Container::linkContentAbsolute(std::string path)
00093 {
00094     YAML::Node node = YAML_FILE::readFile(path);
00095     if(!loadName(node)) return "";
00096
00097     if(node["textures"])
00098     {
00099         loadSprites(node["textures"]);
00100     }
00101
00102     if(node["focus"])
00103     {
00104         loadFocus(node["focus"]);
00105     }
00106
00107     return name;
00108 }
00109 }
00110
00111 bool Container::loadName(YAML::Node node)
00112 {
00113     if(!node["name"])
00114     {
00115         name = "";
00116         return false;
00117     }
00118     name = node["name"].as<std::string>();
00119     return true;
00120 }
00121
00122 void Container::loadSprites(YAML::Node node)
00123 {
00124     for(auto sprite : node)
00125     {
00126         if(!sprite["path"]) continue;
00127         if(!sprite["graphics"]) continue;
00128
00129         std::string path = PASSETS::GRAPHIC_ + sprite["path"].as<std::string>();
00130         Image image = LoadImage(path.c_str());
00131
00132         if(sprite["resize"])
00133         {
00134             int x = image.width * sprite["resize"][0].as<float>();
00135             int y = image.height * sprite["resize"][1].as<float>();
00136             ImageResize(&image, x, y);
00137         }
00138
00139         sprites.emplace_back();
00140         for(auto img : sprite["graphics"])
00141         {
00142             float x, y, w, h;
00143             int repeat = 1;
00144             int gapX = 0;
00145             int gapY = 0;
00146
00147             int dx = 1;
00148             int dy = 1;
00149
00150             if(img["x"])
00151                 x = img["x"].as<float>() / 100.0;
00152             else x = 0;
00153             if(img["y"])
00154                 y = img["y"].as<float>() / 100.0;
00155             else y = 0;
00156             if(img["w"])
00157                 w = img["w"].as<float>() / 100.0;
00158             else w = 1;
00159             if(img["h"])
00160                 h = img["h"].as<float>() / 100.0;
00161             else h = 1;
00162             if(img["repeat"])
00163                 repeat = img["repeat"].as<int>();
00164             if(img["gapX"])
00165                 gapX = img["gapX"].as<int>();
00166             if(img["gapY"])
00167                 gapY = img["gapY"].as<int>();
00168
00169             if(img["dx"])
00170                 dx = img["dx"].as<int>();
00171             if(dx < 0) dx = -1;
00172             else dx = 1;
00173
00174             if(img["dy"])

```



```

00175         dy = img["dy"].as<int>();
00176         if(dy < 0) dy = -1;
00177         else dy = 1;
00178
00179         int imgw = image.width;
00180         int imgh = image.height;
00181
00182         if(img["axis"] && img["axis"].as<std::string>() == "horizontal")
00183         {
00184             for(float j = y; j >= 0 && j + h < 1 + 1e-2; j += dy * (gapY + h))
00185             {
00186                 for(float i = x; i >= 0 && i + w <= 1 + 1e-2 && repeat--; i += dx * (gapX + w))
00187                 {
00188                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00189                     Image img2 = ImageFromImage(image, rect);
00190                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00191                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00192                     sprites.back().push_back(vis);
00193
00194                     UnloadImage(img2);
00195                 }
00196             }
00197         }else
00198         {
00199             for(float i = x; i >= 0 && i + w <= 1 + 1e-2; i += dx * (gapX + w))
00200             {
00201                 for(float j = y; j >= 0 && j + h < 1 + 1e-2 && repeat--; j += dy * (gapY + h))
00202                 {
00203                     Rectangle rect = {i * imgw, j * imgh, w * imgw, h * imgh};
00204                     Image img2 = ImageFromImage(image, rect);
00205                     Texture2D *txt = new Texture2D(LoadTextureFromImage(img2));
00206                     Visual *vis = new Visual(txt, this, {0, 0, 1, 1});
00207                     sprites.back().push_back(vis);
00208
00209                     UnloadImage(img2);
00210                 }
00211             }
00212         }
00213     }
00214     UnloadImage(image);
00215 }
00216 }
00217
00218 void Container::loadFocus(YAML::Node node)
00219 {
00220     focus[0] = node[0].as<int>();
00221     focus[1] = node[1].as<int>();
00222 }
00223
00224 void Container::chooseSprite(int index)
00225 {
00226     if(sprites.empty()) return;
00227     if(index < 0 || index >= sprites.size()) return;
00228     focus[0] = index;
00229 }
00230
00231 void Container::chooseImage(int index)
00232 {
00233     if(sprites.empty()) return;
00234     if(index < 0 || index >= sprites.size()) return;
00235     focus[1] = index;
00236 }
00237
00238 void Container::chooseImage(int index, int index2)
00239 {
00240     if(sprites.empty()) return;
00241     if(index < 0 || index >= sprites.size()) return;
00242     if(index2 < 0 || index2 >= sprites.at(index).size()) return;
00243     focus[0] = index;
00244     focus[1] = index2;
00245 }
00246
00247 void Container::nextImage()
00248 {
00249     if(sprites.empty()) return;
00250     focus[1]++;
00251     if(focus[1] >= sprites.at(focus[0]).size()) focus[1] = 0;
00252 }
00253
00254 void Container::prevImage()
00255 {
00256     if(sprites.empty()) return;
00257     focus[1]--;
00258     if(focus[1] < 0) focus[1] = sprites.at(focus[0]).size() - 1;
00259 }
00260
00261 void Container::nextSprite()

```

```

00262 {
00263     if (sprites.empty()) return;
00264     focus[0]++;
00265     if (focus[0] >= sprites.size()) focus[0] = 0;
00266 }
00267
00268 void Container::prevSprite()
00269 {
00270     if (sprites.empty()) return;
00271     focus[0]--;
00272     if (focus[0] < 0) focus[0] = sprites.size() - 1;
00273 }
00274
00275 std::string Container::getName()
00276 {
00277     return name;
00278 }
00279
00280 void Container::setProbability(int prob)
00281 {
00282     probability = prob;
00283 }
00284
00285 int Container::getProbability()
00286 {
00287     return probability;
00288 }

```

## 8.50 src/frame/src/constructor.cpp File Reference

```
#include <frame.hpp>
```

## 8.51 constructor.cpp

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

```

00001 #include <frame.hpp>
00002
00010 Frame::Frame(Frame* par, Rectangle rel)
00011 {
00012     parent = nullptr;
00013     if (par == nullptr)
00014     {
00015         throw std::runtime_error("Frame::Frame(Frame* par, fRect rel): par is nullptr");
00016         return ;
00017     }
00018     parent = par;
00019     relative[0] = rel.x;
00020     relative[1] = rel.y;
00021     relative[2] = rel.width;
00022     relative[3] = rel.height;
00023
00024     parent->addSubframe(this);
00025
00026     updateFrame();
00027 }
00033 Frame::Frame(Frame* self)
00034 {
00035     parent = nullptr;
00036     if (self == nullptr)
00037     {
00038         throw std::runtime_error("Frame::Frame(Frame* self): self is nullptr");
00039         return ;
00040     }
00041     parent = self->parent;
00042     relative = self->relative;
00043     frame = self->frame;
00044     for (auto& i : self->subframes)
00045     {
00046         subframes.push_back(i);
00047     }
00048 }
00057 Frame::Frame(Rectangle rec)
00058 {
00059     parent = nullptr;

```

```

00060     frame = rec;
00061     parent = nullptr;
00062     relative = {1, 1, 1, 1};
00063 }
00064

```

## 8.52 src/game/src/constructor.cpp File Reference

```

#include "raylib.h"
#include <const/path/atb.hpp>
#include <file.hpp>
#include <vector.hpp>
#include <object.hpp>
#include <chunk.hpp>
#include <game.hpp>

```

## 8.53 constructor.cpp

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

```

00001 #include "raylib.h"
00002 #include <const/path/atb.hpp>
00003 #include <file.hpp>
00004 #include <vector.hpp>
00005 #include <object.hpp>
00006 #include <chunk.hpp>
00007 #include <game.hpp>
00008
00009 Game::Game(Frame* frame, Rectangle rect) : Interface(frame, rect)
00010 {
00011     initState = true;
00012 }
00013
00014 Game::Game(Game* other) : Interface(other)
00015 {
00016     initState = true;
00017 }
00018
00019 Game::Game(Game* other, Rectangle rect) : Interface(other, rect)
00020 {
00021     initState = true;
00022 }
00023
00024 Game::Game(Game* other, Frame* frame, Rectangle rect) : Interface(other, frame, rect)
00025 {
00026     initState = true;
00027 }
00028
00029 std::string Game::linkContentAbsolute(std::string path)
00030 {
00031     YAML::Node node = YAML_FILE::readFile(path);
00032     if(!loadName(node)) return "";
00033
00034     if(node["textures"])
00035         loadSprites(node["textures"]);
00036
00037     if(node["focus"])
00038         loadFocus(node["focus"]);
00039     else chooseImage(0, 0);
00040
00041     if(node["object"])
00042     {
00043         loadObject(node["object"]);
00044         for(int i = 0; i < getContainersSize(); i++)
00045             getContainers(i)->hide();
00046         main = getContainers(0);
00047         main->show();
00048     }
00049     if(node["collide"])
00050         loadCollide(node["collide"]);
00051
00052     if(node["chunk"])

```

```

00053         loadChunk(node["chunk"]);
00054
00055     if(node["attach-object"])
00056         loadAttachObject(node["attach-object"]);
00057
00058     if(node["control"])
00059         loadControl(node["control"]);
00060
00061     if(node["event"])
00062         loadEvent(node["event"]);
00063
00064     if(node["button"])
00065         loadButton(node["button"]);
00066     return getName();
00067 }
00068
00069 void Game::loadCollide(YAML::Node node)
00070 {
00071 }
00072
00073 void Game::loadMap()
00074 {
00075     if(cache.empty()) return ;
00076     while(!chunks.empty())
00077     {
00078         fRect rec = chunks.back()->getRelative();
00079         if(rec[1] > 1) chunks.pop_back();
00080         else break;
00081     }
00082     if(chunks.empty())
00083     {
00084         Rectangle rel;
00085         rel.width = cache[0]->getRelative()[2];
00086         rel.height = cache[0]->getRelative()[3];
00087         rel.x = 0;
00088         rel.y = (1.01 - rel.height);
00089
00090         Chunk* chunk = new Chunk(cache[0], this, rel);
00091         chunks.push_front(chunk);
00092         for(int i = 0; i < 3; i++)
00093         {
00094             rel.y += 0.005 - rel.height;
00095             chunk = new Chunk(cache[0], this, rel);
00096             chunks.push_front(chunk);
00097         }
00098     }
00099     while(chunks.front()->getRelative()[1] > 0)
00100     {
00101         Rectangle rel;
00102         rel.width = chunks.front()->getRelative()[2];
00103         rel.height = chunks.front()->getRelative()[3];
00104         rel.x = 0;
00105         rel.y = (chunks.front()->getRelative()[1] + 0.005 - rel.height);
00106
00107         int id = GetRandomValue(0, cache.size() - 1);
00108         Chunk* chunk = new Chunk(cache[id], this, rel);
00109         chunks.push_front(chunk);
00110     }
00111 }
00112
00113 void Game::loadChunk(YAML::Node node)
00114 {
00115     for(auto i : node)
00116     {
00117         float x = 0, y = 0, w = 1, h = 1;
00118         int repeat = 1;
00119         std::string path = i["file"].as<std::string>();
00120         if(i["x"]) x = i["x"].as<float>() / 100;
00121         if(i["y"]) y = i["y"].as<float>() / 100;
00122         if(i["w"]) w = i["w"].as<float>() / 100;
00123         if(i["h"]) h = i["h"].as<float>() / 100;
00124         if(i["repeat"]) repeat = i["repeat"].as<int>();
00125         fPoint direction = {1, 0};
00126         float velo = 0.002;
00127         if(i["velocity"])
00128         {
00129             velo = i["velocity"][0].as<float>();
00130             direction = {i["velocity"][1].as<float>(), i["velocity"][2].as<float>()};
00131         }
00132         float angle = VECTOR2D::getAngle(direction);
00133         fPoint displacement = {velo * cos(angle), velo * sin(angle)};
00134         Chunk* chunk = new Chunk(this, {x, y, w, h});
00135         chunk->linkContent(path);
00136         chunk->setVelocity(displacement);
00137         cache.push_back(chunk);
00138         while(--repeat > 0)
00139             cache.push_back(new Chunk(cache[0]));

```

```

00140     }
00141 }
00142
00143 void Game::loadAttactObject (YAML::Node node)
00144 {
00145     for(auto i : node)
00146     {
00147         int id = i["chunk"].as<int>();
00148         int objID = i["object"][0].as<int>();
00149         int prob = i["object"][1].as<int>();
00150         Container* container = getContainers(objID);
00151         container->setProbability(prob);
00152         cache[id]->addVisitor(container);
00153     }
00154 }
00155
00156 void Game::loadEvent (YAML::Node node)
00157 {
00158     if(node["map-speed"])
00159     {
00160         mapSpeed = node["map-speed"].as<float>();
00161     }
00162     if(node["map-direction"])
00163     {
00164         mapDirection[0] = node["map-direction"][0].as<float>();
00165         mapDirection[1] = node["map-direction"][1].as<float>();
00166     }
00167     float angle = VECTOR2D::getAngle(mapDirection);
00168     std::cout << "hehe: " << angle << std::endl;
00169     mapDisplacement[0] = mapSpeed * cos(angle);
00170     mapDisplacement[1] = mapSpeed * sin(angle);
00171 }

```

## 8.54 src/interface/src/constructor.cpp File Reference

```

#include "raylib.h"
#include <interface.hpp>
#include <const/path/atb.hpp>
#include <file.hpp>
#include <object.hpp>
#include <chunk.hpp>

```

## 8.55 constructor.cpp

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

```

00001 #include "raylib.h"
00002 #include <interface.hpp>
00003 #include <const/path/atb.hpp>
00004 #include <file.hpp>
00005 #include <object.hpp>
00006 #include <chunk.hpp>
00007
00008 Interface::Interface(Frame* frame, Rectangle rect) : Container(frame, rect)
00009 {
00010 }
00011
00012 Interface::Interface(Interface* other) : Container(other)
00013 {
00014     for(auto i : other->nested)
00015     {
00016         Rectangle rel;
00017         rel.x = i->getRelative()[0];
00018         rel.y = i->getRelative()[1];
00019         rel.width = i->getRelative()[2];
00020         rel.height = i->getRelative()[3];
00021         nested.push_back(new Interface(i, this, rel));
00022     }
00023     for(auto i : other->containers)
00024     {
00025         Rectangle rel;
00026         rel.x = i->getRelative()[0];

```

```

00027         rel.y = i->getRelative()[1];
00028         rel.width = i->getRelative()[2];
00029         rel.height = i->getRelative()[3];
00030         containers.push_back(new Container(i, this, rel));
00031     }
00032 }
00033
00034 Interface::Interface(Interface* other, Rectangle rect) : Container(other, rect)
00035 {
00036     for(auto i : other->nested)
00037     {
00038         Rectangle rel;
00039         rel.x = i->getRelative()[0];
00040         rel.y = i->getRelative()[1];
00041         rel.width = i->getRelative()[2];
00042         rel.height = i->getRelative()[3];
00043         nested.push_back(new Interface(i, this, rel));
00044     }
00045     for(auto i : other->containers)
00046     {
00047         Rectangle rel;
00048         rel.x = i->getRelative()[0];
00049         rel.y = i->getRelative()[1];
00050         rel.width = i->getRelative()[2];
00051         rel.height = i->getRelative()[3];
00052         containers.push_back(new Container(i, this, rel));
00053     }
00054 }
00055
00056 Interface::Interface(Interface* other, Frame* frame, Rectangle rect) : Container(other, frame, rect)
00057 {
00058     for(auto i : other->nested)
00059     {
00060         Rectangle rel;
00061         rel.x = i->getRelative()[0];
00062         rel.y = i->getRelative()[1];
00063         rel.width = i->getRelative()[2];
00064         rel.height = i->getRelative()[3];
00065         nested.push_back(new Interface(i, this, rel));
00066     }
00067     for(auto i : other->containers)
00068     {
00069         Rectangle rel;
00070         rel.x = i->getRelative()[0];
00071         rel.y = i->getRelative()[1];
00072         rel.width = i->getRelative()[2];
00073         rel.height = i->getRelative()[3];
00074         containers.push_back(new Container(i, this, rel));
00075     }
00076 }
00077
00078 std::string Interface::linkContent(std::string path)
00079 {
00080     return linkContentAbsolute(PATB::INTERFACE_ + path);
00081 }
00082
00083 std::string Interface::linkContentAbsolute(std::string path)
00084 {
00085     YAML::Node node = YAML_FILE::readFile(path);
00086     if(!loadName(node)) return "";
00087
00088     if(node["textures"])
00089         loadSprites(node["textures"]);
00090
00091     if(node["focus"])
00092         loadFocus(node["focus"]);
00093     else chooseImage(0, 0);
00094
00095     if(node["object"])
00096         loadObject(node["object"]);
00097
00098     if(node["control"])
00099         loadControl(node["control"]);
00100
00101     if(node["button"])
00102         loadButton(node["button"]);
00103
00104     // if(node["collide"])
00105     //     loadCollide(node["collide"]);
00106
00107     // if(node["chunk"])
00108     //     loadChunk(node["chunk"]);
00109
00110
00111     // if(node["event"])
00112     //     loadEvent(node["event"]);
00113

```

```

00114     return getName();
00115 }
00116
00117 void Interface::loadObject(YAML::Node node)
00118 {
00119     for(auto i : node)
00120     {
00121         Rectangle rel({0, 0, 0, 0});
00122         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00123         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00124         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00125         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00126         Container *obj;
00127         obj = new Object(this, rel);
00128         obj->linkContent(i["path"].as<std::string>());
00129         containers.push_back(obj);
00130     }
00131 }
00132
00133
00134 void Interface::loadControl(YAML::Node node)
00135 {
00136     for(auto stroke : node)
00137     {
00138         KeyStroke* k = new KeyStroke();
00139         for(auto key : stroke["key"])
00140         {
00141             k->add(toKey(key.as<std::string>()));
00142         }
00143         std::string action = stroke["action"].as<std::string>();
00144
00145         if(action == "move-object")
00146         {
00147             int id = stroke["args"][0].as<int>();
00148             float v = stroke["args"][1].as<float>() / 100.0;
00149             float x = stroke["args"][2].as<float>();
00150             float y = stroke["args"][3].as<float>();
00151             moveObjectAction* action = new moveObjectAction(containers[id], fPoint({x, y}), v);
00152             k->addAction(action);
00153         }
00154
00155         keystrokes.push_back(k);
00156     }
00157 }
00158
00159 void Interface::loadButton(YAML::Node node)
00160 {
00161     for(auto i : node)
00162     {
00163         Rectangle rel({0, 0, 0, 0});
00164         if(i["x"]) rel.x = i["x"].as<float>() / 100;
00165         if(i["y"]) rel.y = i["y"].as<float>() / 100;
00166         if(i["w"]) rel.width = i["w"].as<float>() / 100;
00167         if(i["h"]) rel.height = i["h"].as<float>() / 100;
00168         ButtonImage *obj;
00169         obj = new ButtonImage(this, rel);
00170         obj->linkContent(i["path"].as<std::string>());
00171         obj->show();
00172         containers.push_back(obj);
00173     }
00174 }
00175
00176 Container* Interface::getContainers(int id)
00177 {
00178     if(id < 0 || id >= containers.size()) return nullptr;
00179     return containers[id];
00180 }
00181
00182 int Interface::getContainersSize()
00183 {
00184     return containers.size();
00185 }

```

## 8.56 src/object/src/constructor.cpp File Reference

```

#include "container.hpp"
#include <object.hpp>
#include <const/path/atb.hpp>
#include <file.hpp>

```

## 8.57 constructor.cpp

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

```

00001 #include "container.hpp"
00002 #include <object.hpp>
00003 #include <const/path/atb.hpp>
00004 #include <file.hpp>
00005
00006 Object::Object(Frame* f, Rectangle rel) : Container(f, rel)
00007 {
00008     waitUntil = std::chrono::steady_clock::now();
00009 }
00010
00011 Object::Object(Object* other) : Container(other)
00012 {
00013     waitUntil = std::chrono::steady_clock::now();
00014 }
00015
00016 Object::Object(Object* other, Rectangle rel) : Container(other, rel)
00017 {
00018     waitUntil = std::chrono::steady_clock::now();
00019 }
00020
00021 Object::Object(Object* other, Frame* f, Rectangle rel) : Container(other, f, rel)
00022 {
00023     waitUntil = std::chrono::steady_clock::now();
00024 }
00025
00026 std::string Object::linkContent(std::string path)
00027 {
00028     return linkContentAbsolute(PATB::OBJECT_ + path);
00029 }
00030
00031 std::string Object::linkContentAbsolute(std::string path)
00032 {
00033     YAML::Node node = YAML_FILE::readFile(path);
00034     if(!loadName(node)) return "";
00035     if(node["textures"]) loadSprites(node["textures"]);
00036     if(node["control"]) loadControl(node["control"]);
00037
00038     chooseImage(0, 0);
00039     if(node["focus"])
00040         loadFocus(node["focus"]);
00041
00042     return "";
00043 }
00044
00045 void Object::loadControl(YAML::Node node)
00046 {
00047     for(auto stroke : node)
00048     {
00049         strokes.emplace_back();
00050         KeyStroke* k = new KeyStroke();
00051         for(auto key : stroke["key"])
00052         {
00053             k->add(toKey(key.as<std::string>()));
00054         }
00055         for(auto sprite : stroke["sprite"])
00056         {
00057             iPoint p;
00058             int delay = 0;
00059             p[0] = sprite[0].as<int>();
00060             p[1] = sprite[1].as<int>();
00061             if(p.size() >= 3)
00062                 delay = sprite[2].as<int>();
00063             k->addAction(new changeImageAction(this, p));
00064         }
00065         strokes.back().stroke = k;
00066     }
00067 }

```

## 8.58 src/visual/src/constructor.cpp File Reference

```
#include <visual.hpp>
```



## 8.59 constructor.cpp

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

```
00001 #include <visual.hpp>
00002
00003
00004 Visual::Visual(Texture2D* txtr, Frame* frame, Rectangle rect) : Frame(frame, rect)
00005 {
00006     m_texture = std::shared_ptr<Texture2D>(txtr, [](Texture2D* texture){
00007         UnloadTexture(*texture);
00008         texture = nullptr;
00009     });
00010     resizable = true;
00011     fitFrame();
00012 }
00013
00014 Visual::Visual(Visual* visual) : Frame(visual)
00015 {
00016     resizable = false;
00017     m_texture = visual->m_texture;
00018     fitFrame();
00019 }
00020
00021 Visual::Visual(Visual* visual, Rectangle rect) : Frame(visual, rect)
00022 {
00023     resizable = false;
00024     m_texture = visual->m_texture;
00025     fitFrame();
00026 }
00027
00028 Visual::Visual(Visual* visual, Frame* frame, Rectangle rect) : Frame(frame, rect)
00029 {
00030     resizable = false;
00031     m_texture = visual->m_texture;
00032     fitFrame();
00033 }
```

## 8.60 src/window/src/constructor.cpp File Reference

```
#include <window.hpp>
#include <game.hpp>
#include <const/path/atb.hpp>
#include <file.hpp>
```

## 8.61 constructor.cpp

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

```
00001 #include <window.hpp>
00002 #include <game.hpp>
00003 #include <const/path/atb.hpp>
00004
00005 #include <file.hpp>
00006
00007 Window::Window()
00008 {
00009     Wcontent.width = 1200;
00010     Wcontent.height = 668;
00011     Wcontent.title = "Crossy Road clone";
00012     SetConfigFlags(FLAG_WINDOW_RESIZABLE | FLAG_VSYNC_HINT);
00013     InitWindow(Wcontent.width, Wcontent.height, Wcontent.title.c_str());
00014     SetTargetFPS(60);
00015
00016     UI.setRootFrame(new Frame({0, 0, Wcontent.width, Wcontent.height}));
00017 }
00018
00019 Window::Window(std::string path)
00020 {
00021     path = PATB::WINDOW_ + path;
00022     YAML::Node config = YAML_FILE::readFile(path);
00023     initRaylib(config);
00024 }
```

```

00024
00025     loadInterface(config["interface-list"]);
00026     loadGame(config["game"]);
00027     if(config["choose-interface"])
00028         UI.push(config["choose-interface"].as<std::string>());
00029
00030     if(config["input-delay"])
00031     {
00032         double delay = config["input-delay"].as<int>() / 1000.0;
00033         Wcontent.input_delay = std::chrono::duration<double>(delay);
00034     }else
00035     {
00036         Wcontent.input_delay = std::chrono::duration<int>(50) / 1000.0;
00037     }
00038
00039     if(config["runtime-delay"])
00040     {
00041         double delay = config["runtime-delay"].as<int>() / 1000.0;
00042         Wcontent.runtime_delay = std::chrono::duration<double>(delay);
00043     }else
00044     {
00045         Wcontent.runtime_delay = std::chrono::duration<int>(40) / 1000.0;
00046     }
00047 }
00048
00049
00050 void Window::initRaylib(YAML::Node config)
00051 {
00052     Wcontent.width = config["width"].as<int>();
00053     Wcontent.height = config["height"].as<int>();
00054     Wcontent.title = config["title"].as<std::string>();
00055     // enable resizable window and vsync
00056
00057     SetConfigFlags(FLAG_WINDOW_RESIZABLE | FLAG_VSYNC_HINT);
00058     InitWindow(Wcontent.width, Wcontent.height, Wcontent.title.c_str());
00059     SetTargetFPS(60);
00060     Wcontent.setStatus(true);
00061     UI.setRootFrame(new Frame({0, 0, Wcontent.width, Wcontent.height}));
00062 }
00063
00064 void Window::loadInterface(YAML::Node node)
00065 {
00066     if(!node) return ;
00067
00068     UI.setInterfacePool(new InterfacePool());
00069     for(auto i : node)
00070     {
00071         std::string path = i["file"].as<std::string>();
00072         float x = 0, y = 0, w = 1, h = 1;
00073
00074         if(i["x"]) x = i["x"].as<float>() / 100;
00075         if(i["y"]) y = i["y"].as<float>() / 100;
00076         if(i["w"]) w = i["w"].as<float>() / 100;
00077         if(i["h"]) h = i["h"].as<float>() / 100;
00078
00079         Interface* inf = new Interface(UI.getRootFrame(), {x, y, w, h});
00080         inf->linkContent(path);
00081         UI.load(inf);
00082     }
00083 }
00084
00085
00086 void Window::loadGame(YAML::Node node)
00087 {
00088     if(!node) return ;
00089
00090     std::string path = node["file"].as<std::string>();
00091     float x = 0, y = 0, w = 1, h = 1;
00092
00093     if(node["x"]) x = node["x"].as<float>() / 100;
00094     if(node["y"]) y = node["y"].as<float>() / 100;
00095     if(node["w"]) w = node["w"].as<float>() / 100;
00096     if(node["h"]) h = node["h"].as<float>() / 100;
00097
00098     Interface* inf = new Game(UI.getRootFrame(), {x, y, w, h});
00099     inf->linkContent(path);
00100     UI.load(inf);
00101 }
00102 }

```

## 8.62 src/button/src/destructor.cpp File Reference

```
#include <button.hpp>
```

## 8.63 destructor.cpp

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

```
00001 #include <button.hpp>
00002
00003 ButtonImage::~ButtonImage()
00004 {
00005     for(auto &action : actions)
00006     {
00007         delete action;
00008     }
00009 }
```

## 8.64 src/chunk/src/destructor.cpp File Reference

```
#include <chunk.hpp>
```

## 8.65 destructor.cpp

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

```
00001 #include <chunk.hpp>
00002
00003 Chunk::~Chunk()
00004 {
00005     for(auto i : visitor)
00006         delete i;
00007
00008     while(!Entity.empty())
00009     {
00010         delete Entity.back();
00011         Entity.pop_back();
00012     }
00013 }
00014
```

## 8.66 src/container/src/destructor.cpp File Reference

```
#include <container.hpp>
```

## 8.67 destructor.cpp

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

```
00001 #include <container.hpp>
00002
00003 Container::~Container()
00004 {
00005     for(Sprite & sprite : sprites)
00006     {
00007         for(auto& frame : sprite)
00008             delete frame;
00009         sprite.clear();
00010     }
00011 }
```

## 8.68 src/frame/src/destructor.cpp File Reference

```
#include <frame.hpp>
```

### 8.69 destructor.cpp

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

```
00001 #include <frame.hpp>
00002
00010 Frame::~Frame()
00011 {
00012 }
```

## 8.70 src/game/src/destructor.cpp File Reference

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

### 8.71 destructor.cpp

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

```
00001 #include <game.hpp>
00002
00003
00004 Game::~Game()
00005 {
00006     for(auto &i : cache)
00007     {
00008         delete i;
00009     }
00010
00011     for(auto &i : chunks)
00012     {
00013         delete i;
00014     }
00015 }
```

## 8.72 src/interface/src/destructor.cpp File Reference

```
#include <interface.hpp>
```

### 8.73 destructor.cpp

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

```
00001 #include <interface.hpp>
00002
00003
00004 Interface::~Interface()
00005 {
00006     for (auto& i : containers)
00007         delete i;
00008     containers.clear();
00009
00010     for (auto& i : nested)
00011         delete i;
00012     nested.clear();
00013
00014     for (auto& i : keystrokes)
00015         delete i;
00016 }
```

## 8.74 src/object/src/destructor.cpp File Reference

```
#include <object.hpp>
```

## 8.75 destructor.cpp

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

```
00001 #include <object.hpp>
00002
00003
00004 Object::~Object ()
00005 {
00006     for (auto &stroke : strokes)
00007     {
00008         delete stroke.stroke;
00009     }
00010     strokes.clear();
00011
00012 }
```

## 8.76 src/visual/src/destructor.cpp File Reference

```
#include <visual.hpp>
```

### Functions

- void [deleteSprite](#) ([Sprite](#) sprite)
- void [deleteSprites](#) (std::vector< [Sprite](#) > \*&sprites)

### 8.76.1 Function Documentation

#### 8.76.1.1 deleteSprite()

```
void deleteSprite (
    Sprite sprite )
```

Definition at line 17 of file [destructor.cpp](#).

```
00018 {
00019     for(auto& frame : sprite)
00020         delete frame;
00021     sprite.clear();
00022 }
```

#### 8.76.1.2 deleteSprites()

```
void deleteSprites (
    std::vector< Sprite > *& sprites )
```

Definition at line 24 of file [destructor.cpp](#).

```
00025 {
00026     for(Sprite & sprite : *sprites)
00027     {
00028         deleteSprite(sprite);
00029     }
00030     delete sprites;
00031 }
```

## 8.77 destructor.cpp

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

```
00001 #include <visual.hpp>
00002
00003 void Visual::deleteTexture2D(Texture2D*& texture)
00004 {
00005     if(texture != nullptr)
00006     {
00007         UnloadTexture(*texture);
00008         texture = nullptr;
00009     }
00010 }
00011
00012 Visual::~Visual()
00013 {
00014     m_texture.reset();
00015 }
00016
00017 void deleteSprite(Sprite sprite)
00018 {
00019     for(auto& frame : sprite)
00020         delete frame;
00021     sprite.clear();
00022 }
00023
00024 void deleteSprites(std::vector<Sprite*>& sprites)
00025 {
00026     for(Sprite & sprite : *sprites)
00027     {
00028         deleteSprite(sprite);
00029     }
00030     delete sprites;
00031 }
```

## 8.78 src/window/src/destructor.cpp File Reference

```
#include <window.hpp>
```

## 8.79 destructor.cpp

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

```
00001 #include <window.hpp>
00002
00003 Window::~Window()
00004 {
00005     CloseWindow();
00006 }
```

## 8.80 src/chunk/include/chunk.hpp File Reference

```
#include "action.hpp"
#include <chrono>
#include <deque>
#include <interface.hpp>
```

### Classes

- class [Chunk](#)  
*manages the spawning of chunks and how entities interact with them*
- class [moveEntityAction](#)

## 8.81 chunk.hpp

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

```
00001 #ifndef CHUNK_HPP
00002 #define CHUNK_HPP
00003
00004 #include "action.hpp"
00005 #include <chrono>
00006 #include <deque>
00007
00008 #include <interface.hpp>
00009
00016 class Chunk : public Interface
00017 {
00018 private:
00019     friend class moveEntityAction;
00020     fPoint velocity;
00021     std::vector<Container*> visiter;
00022     std::deque<Container*> Entity;
00023     std::chrono::time_point<std::chrono::system_clock> spawnClock;
00024     std::chrono::time_point<std::chrono::system_clock> moveClock;
00025     constexpr static std::chrono::duration<double> spawnTime = std::chrono::duration<double>(1.0);
00026     constexpr static std::chrono::duration<double> moveTime = std::chrono::duration<double>(0.1);
00027
00028 protected:
00029     void drawEntity();
00030     Container* randomEntity();
00031     void movingEntity();
00032 public:
00033     Chunk(Frame*, Rectangle);
00034     Chunk(Chunk*);
00035     Chunk(Chunk*, Rectangle);
00036     Chunk(Chunk*, Frame*, Rectangle);
00037     ~Chunk();
00038
00039     void addVisiter(Container*);
00040     void addVisiter(Container*, int);
00041     void addVisiter(Container*, Rectangle);
00042     void addVisiter(Container*, int, Rectangle);
00043     void generateEntity();
00044
00045     void setVelocity(fPoint);
00046
00047     std::string linkContent(std::string path) override;
00048     Action* getRuntimeEvent() override;
00049
00050     void draw() override;
00051 };
00052
00053 class moveEntityAction : public Action
00054 {
00055 private:
00056     Chunk* chunk;
00057 public:
00058     moveEntityAction(Chunk*);
00059     ~moveEntityAction();
00060
00061     void execute() override;
00062     Action* clone() override;
00063 };
00064 #endif
00065
```

## 8.82 src/chunk/src/action/moveentity.cpp File Reference

```
#include <chunk.hpp>
```

## 8.83 moveentity.cpp

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

```
00001 #include <chunk.hpp>
```

```

00002
00003 moveEntityAction::moveEntityAction(Chunk* chunk) : chunk(chunk)
00004 {
00005 }
00006
00007 moveEntityAction::~moveEntityAction()
00008 {
00009 }
00010
00011 void moveEntityAction::execute()
00012 {
00013     chunk->movingEntity();
00014 }
00015
00016 Action* moveEntityAction::clone()
00017 {
00018     return new moveEntityAction(chunk);
00019 }

```

## 8.84 src/container/include/container.hpp File Reference

```

#include <vector>
#include <visual.hpp>
#include <frame.hpp>
#include <action.hpp>
#include <const/datatype.hpp>
#include <const/path/atb.hpp>
#include <file.hpp>

```

### Classes

- class [Container](#)  
*holds specific entities and their behavior*
- class [changeImageAction](#)  
*changes display image of container*

## 8.85 container.hpp

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

```

00001 #ifndef CONTAINER_HPP
00002 #define CONTAINER_HPP
00003
00004 #include <vector>
00005
00006 #include <visual.hpp>
00007 #include <frame.hpp>
00008 #include <action.hpp>
00009 #include <const/datatype.hpp>
00010 #include <const/path/atb.hpp>
00011 #include <file.hpp>
00012
00019 class Container : public Frame
00020 {
00021 private:
00022     friend class changeImageAction;
00023     static int id_count;
00024     int instance_id;
00025     int probability;
00026
00027     std::vector<Sprite> sprites;
00028     std::string name;
00029     iPoint focus;
00030     bool visible;
00031
00032 protected:

```



```

00033     bool loadName(YAML::Node node);
00034     void loadSprites(YAML::Node node);
00035     void loadFocus(YAML::Node node);
00036 public:
00037     Container(Frame*, Rectangle);
00038     Container(Container*);
00039     Container(Container*, Rectangle);
00040     Container(Container*, Frame*, Rectangle);
00041     virtual ~Container();
00042
00043     virtual std::string linkContent(std::string);
00044     virtual std::string linkContentAbsolute(std::string);
00045     std::string getName();
00046
00047     void setProbability(int);
00048     int getProbability();
00049
00053     void chooseSprite(int);
00054
00058     void chooseImage(int);
00059
00063     void chooseImage(int, int);
00064
00068     void nextImage();
00069
00073     void prevImage();
00074
00078     void nextSprite();
00079
00083     void prevSprite();
00084
00085     bool isOverlapping(fPoint);
00086     bool isOverlapping(Rectangle);
00087     bool isOverlapping(Container*);
00088     float OverlappingArea(Rectangle);
00089     float OverlappingArea(Container*);
00090
00091     virtual void draw();
00092     void show();
00093     void hide();
00094     void toggleVisibility();
00095     bool isVisible();
00096     int getInstanceId();
00097
00098     virtual Action* react();
00099     virtual Action* getRuntimeEvent();
00100 };
00101
00108 class changeImageAction : public Action
00109 {
00110 private:
00111     Container* container;
00112     iPoint focus;
00113 public:
00114     changeImageAction(Container*, iPoint);
00115     changeImageAction(changeImageAction*);
00116     ~changeImageAction();
00117     void execute() override;
00118     Action* clone() override;
00119 };
00120 #endif

```

## 8.86 src/container/src/action/changesprite.cpp File Reference

```
#include <container.hpp>
```

## 8.87 changesprite.cpp

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

```

00001 #include <container.hpp>
00002
00003 changeImageAction::changeImageAction(Container* c, iPoint p)
00004 {
00005     container = c;

```

```

00006     focus = p;
00007 }
00008
00009 changeImageAction::changeImageAction(changeImageAction* c)
00010 {
00011     container = c->container;
00012     focus = c->focus;
00013 }
00014
00015 changeImageAction::~changeImageAction()
00016 {
00017     container = nullptr;
00018 }
00019
00020 void changeImageAction::execute()
00021 {
00022     container->chooseImage(focus[0], focus[1]);
00023 }
00024
00025 Action* changeImageAction::clone()
00026 {
00027     return new changeImageAction(this);
00028 }

```

## 8.88 src/container/src/overlap.cpp File Reference

```
#include <container.hpp>
```

## 8.89 overlap.cpp

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

```

00001 #include <container.hpp>
00002
00003 bool Container::isOverlapping(fPoint point)
00004 {
00005     Rectangle rec = getFrame();
00006     return (point[0] >= rec.x && point[0] <= rec.x + rec.width && point[1] >= rec.y && point[1] <=
rec.y + rec.height);
00007 }
00008 }
00009
00010 bool Container::isOverlapping(Rectangle rec)
00011 {
00012     Rectangle rec2 = getFrame();
00013     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
rec2.height && rec.y + rec.height >= rec2.y);
00014 }
00015
00016 bool Container::isOverlapping(Container* container)
00017 {
00018     Rectangle rec = getFrame();
00019     Rectangle rec2 = container->getFrame();
00020     return (rec.x <= rec2.x + rec2.width && rec.x + rec.width >= rec2.x && rec.y <= rec2.y +
rec2.height && rec.y + rec.height >= rec2.y);
00021 }
00022
00023 float Container::OverlappingArea(Rectangle rec)
00024 {
00025     Rectangle rec2 = getFrame();
00026     float x = std::max(rec.x, rec2.x);
00027     float y = std::max(rec.y, rec2.y);
00028     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;
00029     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00030     if(w < 0 || h < 0) return 0;
00031     return w * h;
00032 }
00033
00034 float Container::OverlappingArea(Container* container)
00035 {
00036     Rectangle rec = container->getFrame();
00037     Rectangle rec2 = getFrame();
00038     float x = std::max(rec.x, rec2.x);
00039     float y = std::max(rec.y, rec2.y);
00040     float w = std::min(rec.x + rec.width, rec2.x + rec2.width) - x;

```

```

00041     float h = std::min(rec.y + rec.height, rec2.y + rec2.height) - y;
00042     if(w < 0 || h < 0) return 0;
00043     return w * h;
00044 }

```

## 8.90 src/frame/include/frame.hpp File Reference

```

#include <iostream>
#include <vector>
#include <string>
#include <mutex>
#include <raylib.h>
#include <const/datatype.hpp>

```

### Classes

- class [Frame](#)

*position and size of object on screen*

## 8.91 frame.hpp

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

```

00001 #ifndef FRAME_HPP
00002 #define FRAME_HPP
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <string>
00007 #include <mutex>
00008
00009 #include <raylib.h>
00010
00011 #include <const/datatype.hpp>
00012
00024 class Frame
00025 {
00026 private:
00027     Rectangle frame;
00028     std::vector<Frame*> subframes;
00029     Frame* parent;
00030
00031     fRect relative;
00032
00033     mutable std::mutex mtx;
00034 protected:
00035     virtual void updateFrame(bool recursive = false);
00036     bool isroot() const;
00037     void addSubframe(Frame* subframe);
00038     void removeSubframe(Frame* subframe);
00039
00040     void beginUpdate();
00041     void endUpdate();
00042 public:
00043     Frame(Frame* par, Rectangle rel);
00044     Frame(Frame* self);
00045     Frame(Rectangle rec);
00046     ~Frame();
00047
00048     void plug(Frame* par, fRect rel);
00049     void plug(Frame* par);
00050     void unplug();
00051
00052     void moveTo(fPoint rel);
00053     void moveTo(int x, int y);
00054
00055     void moveCenterTo(fPoint rel);

```

```

00056     void moveCenterTo(int x, int y);
00057
00058     void moveBy(fPoint rel);
00059     void moveBy(int, int);
00060
00061     void resize(fPoint rel);
00062     void resize(int w, int h);
00063
00064     const Rectangle& getFrame() const;
00065     const fRect& getRelative() const;
00066     Frame* getParent();
00067
00068     void setRelative(fRect rel);
00069
00070     const fPoint& getCenter() const;
00071
00072     const float& getX() const;
00073     const float& getY() const;
00074     const float& getW() const;
00075     const float& getH() const;
00076
00077     operator Rectangle() const;
00078     operator fRect() const;
00079     operator iRect() const;
00080
00081 };
00082
00083 #endif

```

## 8.92 src/frame/src/family.cpp File Reference

```

#include <frame.hpp>
#include <algorithm>

```

## 8.93 family.cpp

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

```

00001 #include <frame.hpp>
00002 #include <algorithm>
00003
00004
00012 void Frame::plug(Frame* par, fRect rel)
00013 {
00014     if(par == nullptr)
00015     {
00016         throw std::runtime_error("Frame::plug(Frame* par, fRect rel): par is nullptr");
00017         return ;
00018     }
00019     mtx.lock();
00020     parent = par;
00021     relative = rel;
00022     mtx.unlock();
00023     updateFrame();
00024
00025     parent->addSubframe(this);
00026 }
00027
00034 void Frame::plug(Frame* par)
00035 {
00036     if(par == nullptr)
00037     {
00038         throw std::runtime_error("Frame::plug(Frame* par): par is nullptr");
00039         return ;
00040     }
00041     mtx.lock();
00042     parent = par;
00043     mtx.unlock();
00044     updateFrame();
00045
00046     parent->addSubframe(this);
00047 }
00048
00053 void Frame::unplug()

```

```

00054 {
00055     if(isroot()) return ;
00056     mtx.lock();
00057     parent->removeSubframe(this);
00058     parent = nullptr;
00059     mtx.unlock();
00060 }
00061
00070 void Frame::addSubframe(Frame* subframe)
00071 {
00072     mtx.lock();
00073     subframes.push_back(subframe);
00074     mtx.unlock();
00075 }
00076
00085 void Frame::removeSubframe(Frame* subframe)
00086 {
00087     mtx.lock();
00088     int i = subframes.size() - 1;
00089     while(i >= 0 && subframes.size())
00090     {
00091         while(!subframes.empty() && subframes.back() == subframe)
00092             subframes.pop_back();
00093         i = std::min(i, (int) subframes.size() - 1);
00094         if(!subframes.empty() && subframes[i] == subframe)
00095         {
00096             subframes[i] = subframes.back();
00097             subframes.pop_back();
00098         }
00099     }
00100     mtx.unlock();
00101 }
00102
00107 bool Frame::isroot() const
00108 {
00109     std::lock_guard<std::mutex> lock(mtx);
00110     return parent == nullptr;
00111 }
00112
00113 void Frame::beginUpdate()
00114 {
00115     mtx.lock();
00116 }
00117
00118 void Frame::endUpdate()
00119 {
00120     mtx.unlock();
00121 }

```

## 8.94 src/game/include/game.hpp File Reference

```

#include "action.hpp"
#include <deque>
#include <chrono>
#include <raylib.h>
#include <frame.hpp>
#include <container.hpp>
#include <keystroke.hpp>
#include <interface.hpp>
#include <chunk.hpp>

```

### Classes

- class [Game](#)
- class [moveChunksAction](#)

## 8.95 game.hpp

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

```

00001 #ifndef GAME_HPP
00002 #define GAME_HPP
00003
00004 #include "action.hpp"
00005 #include <deque>
00006 #include <chrono>
00007
00008 #include <raylib.h>
00009
00010 #include <frame.hpp>
00011 #include <container.hpp>
00012 #include <keystroke.hpp>
00013 #include <interface.hpp>
00014 #include <chunk.hpp>
00015
00016 class Game : public Interface
00017 {
00018 private:
00019     friend class moveChunksAction;
00020     std::deque<Chunk*> chunks;
00021     std::vector<Chunk*> cache;
00022     Container* main;
00023     fPoint mapDisplacement;
00024     fPoint mapDirection;
00025     float mapSpeed;
00026     std::chrono::time_point<std::chrono::system_clock> mapSpeedClock;
00027     bool initState;
00028 protected:
00029     void loadChunk(YAML::Node);
00030     void loadCollide(YAML::Node);
00031     void loadEvent(YAML::Node);
00032     void loadAttactObject(YAML::Node);
00033     void loadMap();
00034 public:
00035     Game(Frame*, Rectangle);
00036     Game(Game*);
00037     Game(Game*, Rectangle);
00038     Game(Game*, Frame*, Rectangle);
00039
00040     ~Game();
00041
00042     std::string linkContentAbsolute(std::string path) override;
00043
00044     Action* react() override;
00045     Action* getRuntimeEvent() override;
00046     void draw() override;
00047 };
00048
00049
00050 class moveChunksAction : public Action
00051 {
00052 private:
00053     Game* game;
00054     fPoint delta;
00055     fPoint direction;
00056     float speed;
00057 public:
00058     moveChunksAction(Game*, fPoint);
00059     moveChunksAction(Game*, fPoint, float);
00060     ~moveChunksAction();
00061
00062     void execute() override;
00063     Action* clone() override;
00064 };
00065 #endif

```

## 8.96 src/game/src/action/movechunk.cpp File Reference

```

#include <game.hpp>
#include <vector.hpp>

```

## 8.97 movechunk.cpp

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

```
00001 #include <game.hpp>
00002 #include <vector.hpp>
00003
00004 moveChunksAction::moveChunksAction(Game* game, fPoint delta)
00005 {
00006     this->game = game;
00007     this->delta = delta;
00008 }
00009
00010 moveChunksAction::moveChunksAction(Game* game, fPoint d, float v)
00011 {
00012     this->game = game;
00013     this->direction = d;
00014     this->speed = v;
00015
00016     float angle = VECTOR2D::getAngle(direction);
00017     delta[0] = cos(angle) * speed;
00018     delta[1] = sin(angle) * speed;
00019 }
00020
00021 moveChunksAction::~moveChunksAction()
00022 {
00023 }
00024
00025 void moveChunksAction::execute()
00026 {
00027     for(auto& chunk : game->chunks)
00028     {
00029         chunk->moveBy(delta);
00030     }
00031     game->loadMap();
00032 }
00033
00034 Action* moveChunksAction::clone()
00035 {
00036     return new moveChunksAction(game, delta);
00037 }
```

## 8.98 src/interface/include/interface.hpp File Reference

```
#include "action.hpp"
#include <raylib.h>
#include <frame.hpp>
#include <container.hpp>
#include <keystroke.hpp>
#include <button.hpp>
```

### Classes

- class [Interface](#)  
*where user can interact with the game*
- class [moveObjectAction](#)  
*manages the features of a movement, including which object, speed, direction etc.*

## 8.99 interface.hpp

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

```
00001 #ifndef INTERFACE_HPP
00002 #define INTERFACE_HPP
00003
```

```

00004 #include "action.hpp"
00005 #include <raylib.h>
00006
00007 #include <frame.hpp>
00008 #include <container.hpp>
00009 #include <keystroke.hpp>
00010 #include <button.hpp>
00011
00020 class Interface : public Container
00021 {
00022 private:
00023     friend class moveObjectAction;
00024
00025     std::vector<Container*> containers;
00026     std::vector<Interface*> nested;
00027     std::vector<KeyStroke*> keystrokes;
00028 protected:
00029     void loadObject(YAML::Node);
00030     void loadControl(YAML::Node);
00031     void loadButton(YAML::Node);
00032     void drawNested();
00033     void drawContainers();
00034
00035 public:
00036     Interface(Frame*, Rectangle);
00037     Interface(Interface*);
00038     Interface(Interface*, Rectangle);
00039     Interface(Interface*, Frame*, Rectangle);
00040
00041     ~Interface();
00042
00043     Container* getContainers(int);
00044     int getContainersSize();
00045
00046     std::string linkContent(std::string path) override;
00047     std::string linkContentAbsolute(std::string path) override;
00048
00049     Action* react() override;
00050     Action* getRuntimeEvent() override;
00051     void draw() override;
00052 };
00053
00060 class moveObjectAction : public Action
00061 {
00062 private:
00063     Container* obj;
00064     fPoint delta;
00065     fPoint dir;
00066     float speed;
00067 public:
00068     moveObjectAction(Container* obj, fPoint delta);
00069     moveObjectAction(Container* obj, fPoint dir, float speed);
00070     ~moveObjectAction();
00071
00072     void execute() override;
00073     Action* clone() override;
00074 };
00075 #endif

```

## 8.100 src/interface/src/action/moveobject.cpp File Reference

```

#include <interface.hpp>
#include <vector.hpp>

```

## 8.101 moveobject.cpp

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

```

00001 #include <interface.hpp>
00002 #include <vector.hpp>
00003
00004 moveObjectAction::moveObjectAction(Container* obj, fPoint delta) : obj(obj), delta(delta)
00005 {
00006 }

```



```

00007
00008 moveObjectAction::moveObjectAction(Container* obj, fPoint dir, float speed) : obj(obj), dir(dir),
    speed(speed)
00009 {
00010     float angle = VECTOR2D::getAngle(dir);
00011     delta[0] = cos(angle) * speed;
00012     delta[1] = sin(angle) * speed;
00013 }
00014
00015 moveObjectAction::~moveObjectAction()
00016 {
00017 }
00018
00019 void moveObjectAction::execute()
00020 {
00021     obj->moveBy(delta);
00022 }
00023
00024
00025 Action* moveObjectAction::clone()
00026 {
00027     return new moveObjectAction(obj, delta);
00028 }

```

## 8.102 src/main.cpp File Reference

```

#include <iostream>
#include <window.hpp>

```

### Functions

- int [main](#) ()

### 8.102.1 Function Documentation

#### 8.102.1.1 main()

```
int main ( )
```

Definition at line 5 of file [main.cpp](#).

```

00006 {
00007     Window win("window.yaml");
00008     win.run();
00009     return 0;
00010 }

```

## 8.103 main.cpp

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

```

00001 #include <iostream>
00002
00003 #include <window.hpp>
00004
00005 int main()
00006 {
00007     Window win("window.yaml");
00008     win.run();
00009     return 0;
00010 }

```

## 8.104 src/object/include/object.hpp File Reference

```
#include "action.hpp"
#include <container.hpp>
#include <keystroke.hpp>
#include <chrono>
```

### Classes

- class [Object](#)

## 8.105 object.hpp

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

```
00001 #ifndef OBJECT_HPP
00002 #define OBJECT_HPP
00003
00004 #include "action.hpp"
00005 #include <container.hpp>
00006 #include <keystroke.hpp>
00007 #include <chrono>
00008 class Object : public Container
00009 {
00010 private:
00011     struct ObjectKeyStroke
00012     {
00013         KeyStroke* stroke;
00014     };
00015     std::vector<ObjectKeyStroke> strokes;
00016     std::chrono::time_point<std::chrono::steady_clock> waitUntil;
00017
00018
00019 protected:
00020     void loadControl(YAML::Node node);
00021
00022 public:
00023     Object(Frame*, Rectangle);
00024     Object(Object*);
00025     Object(Object*, Rectangle);
00026     Object(Object*, Frame*, Rectangle);
00027     ~Object();
00028
00029     virtual std::string linkContent(std::string) override;
00030     virtual std::string linkContentAbsolute(std::string) override;
00031
00032     virtual Action* react() override;
00033     void draw() override;
00034 };
00035
00036 #endif
```

## 8.106 src/utils/include/const/datatype.hpp File Reference

```
#include <array>
#include <vector>
#include <string>
```

## Typedefs

- using `iPoint` = `std::array<int, 2>`
- using `fPoint` = `std::array<float, 2>`
- using `iRect` = `std::array<int, 4>`
- using `fRect` = `std::array<float, 4>`
- using `vi` = `std::vector<int>`
- using `vf` = `std::vector<float>`

## 8.106.1 Typedef Documentation

### 8.106.1.1 fPoint

```
using fPoint = std::array<float, 2>
```

Definition at line 9 of file [datatype.hpp](#).

### 8.106.1.2 fRect

```
using fRect = std::array<float, 4>
```

Definition at line 12 of file [datatype.hpp](#).

### 8.106.1.3 iPoint

```
using iPoint = std::array<int, 2>
```

Definition at line 8 of file [datatype.hpp](#).

### 8.106.1.4 iRect

```
using iRect = std::array<int, 4>
```

Definition at line 11 of file [datatype.hpp](#).

### 8.106.1.5 vf

```
using vf = std::vector<float>
```

Definition at line 15 of file [datatype.hpp](#).

### 8.106.1.6 vi

```
using vi = std::vector<int>
```

Definition at line 14 of file [datatype.hpp](#).

## 8.107 datatype.hpp

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

```
00001 #ifndef CONSTANT_HPP
00002 #define CONSTANT_HPP
00003
00004 #include <array>
00005 #include <vector>
00006 #include <string>
00007
00008 using iPoint = std::array<int, 2>;
00009 using fPoint = std::array<float, 2>;
00010
00011 using iRect = std::array<int, 4>;
00012 using fRect = std::array<float, 4>;
00013
00014 using vi = std::vector<int>;
00015 using vf = std::vector<float>;
00016
00017
00018 #endif
```

## 8.108 src/utils/include/const/path/assets.hpp File Reference

```
#include <string>
```

### Namespaces

- namespace [PASSETS](#)

### Variables

- const std::string [PASSETS::GRAPHIC\\_](#) = "assets/graphics/"
- const std::string [PASSETS::SOUND\\_](#) = "assets/sounds/"
- const std::string [PASSETS::FONT\\_](#) = "assets/fonts/"

## 8.109 assets.hpp

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

```
00001 #ifndef ASSETS_HPP
00002 #define ASSETS_HPP
00003
00004 #include <string>
00005
00006 namespace PASSETS
00007 {
00008     extern const std::string GRAPHIC_;
00009     extern const std::string SOUND_;
00010     extern const std::string FONT_;
00011 }
00012
00013 #endif
```

## 8.110 src/utils/include/const/path/atb.hpp File Reference

```
#include <string>
```

## Namespaces

- namespace [PATB](#)

## Variables

- const std::string [PATB::ATB\\_](#) = "atb/"
- const std::string [PATB::WINDOW\\_](#) = "atb/window/"
- const std::string [PATB::INTERFACE\\_](#) = "atb/interface/"
- const std::string [PATB::BUTTON\\_](#) = "atb/button/"
- const std::string [PATB::CONTAINER\\_](#) = "atb/container/"
- const std::string [PATB::OBJECT\\_](#) = "atb/object/"
- const std::string [PATB::MAP\\_](#) = "atb/map/"
- const std::string [PATB::CHUNK\\_](#) = "atb/chunk/"
- const std::string [PATB::BLOCK\\_](#) = "atb/block/"
- const std::string [PATB::ENTITY\\_](#) = "atb/entity/"

## 8.111 atb.hpp

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

```
00001 #ifndef ATB_HPP
00002 #define ATB_HPP
00003
00004 #include <string>
00005
00006 namespace PATB
00007 {
00008     extern const std::string ATB_;
00009
00010     extern const std::string WINDOW_;
00011
00012     extern const std::string INTERFACE_;
00013     extern const std::string BUTTON_;
00014     extern const std::string CONTAINER_;
00015     extern const std::string OBJECT_;
00016
00017
00018     extern const std::string MAP_;
00019     extern const std::string CHUNK_;
00020     extern const std::string BLOCK_;
00021     extern const std::string ENTITY_;
00022 }
00023
00024
00025 #endif
```

## 8.112 src/utils/include/countdown.hpp File Reference

```
#include <chrono>
```

## Classes

- class [CountDown](#)  
*count the time a playthrough takes*

## 8.113 countdown.hpp

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

```
00001 #ifndef COUNT_DOWN_HPP
00002 #define COUNT_DOWN_HPP
00003
00004 #include <chrono>
00005
00012 class Countdown
00013 {
00014 private:
00015     std::chrono::time_point<std::chrono::system_clock> start;
00016     std::chrono::duration<double> elapsed_seconds;
00017     bool finished;
00018 public:
00019     Countdown(int milliseconds);
00020     ~Countdown();
00021     int get();
00022     bool isFinished();
00023     void run();
00024 };
00025
00026 #endif
```

## 8.114 src/utils/include/file.hpp File Reference

```
#include <string>
#include <vector>
#include <yaml-cpp/yaml.h>
```

### Namespaces

- namespace [YAML\\_FILE](#)  
*opens and interacts with YAML files*

### Functions

- bool [YAML\\_FILE::isFile](#) (std::string path)
- YAML::Node [YAML\\_FILE::readFile](#) (std::string path)
- bool [YAML\\_FILE::writeFile](#) (std::string path, YAML::Node content)

## 8.115 file.hpp

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

```
00001 #ifndef UTILS_FILE_H
00002 #define UTILS_FILE_H
00003
00004 #include <string>
00005 #include <vector>
00006
00007 #include <yaml-cpp/yaml.h>
00008
00015 namespace YAML_FILE
00016 {
00017     bool isFile(std::string path);
00018     YAML::Node readFile(std::string path);
00019     bool writeFile(std::string path, YAML::Node content);
00020 }
00021
00022 #endif
```

## 8.116 src/utls/include/keystroke.hpp File Reference

```
#include <action.hpp>
#include <vector>
#include <string>
```

### Classes

- class [KeyStroke](#)  
*manages the link between a key and the actions it performs*

### Functions

- int [toKey](#) (std::string)

## 8.116.1 Function Documentation

### 8.116.1.1 toKey()

```
int toKey (
    std::string x )
```

Definition at line 68 of file [keystroke.cpp](#).

```
00069 {
00070     if(x.size() == 1)
00071     {
00072         if(x[0] >= 'a' && x[0] <= 'z')
00073             return x[0] - 'a' + KEY_A;
00074         if(x[0] >= 'A' && x[0] <= 'Z')
00075             return x[0] - 'A' + KEY_A;
00076
00077         if(x[0] >= '0' && x[0] <= '9')
00078             return x[0] - '0' + KEY_ZERO;
00079
00080         switch (x[0]) {
00081             case ' ':
00082                 return KEY_SPACE;
00083             case '.':
00084                 return KEY_PERIOD;
00085             case ',':
00086                 return KEY_COMMA;
00087             case ';':
00088                 return KEY_SEMICOLON;
00089             case '"':
00090                 return KEY_APOSTROPHE;
00091             case '/':
00092                 return KEY_SLASH;
00093             case '\\':
00094                 return KEY_BACKSLASH;
00095             case '-':
00096                 return KEY_MINUS;
00097             case '=':
00098                 return KEY_EQUAL;
00099             case '[':
00100                 return KEY_LEFT_BRACKET;
00101             case ']':
00102                 return KEY_RIGHT_BRACKET;
00103             case '`':
00104                 return KEY_GRAVE;
00105             case '~':
00106                 return KEY_GRAVE;
00107             case '!':
00108                 return KEY_ONE;
00109             case '@':
00110                 return KEY_ONE;
00111             case '#':
```

```

00112         return KEY_THREE;
00113     case '$':
00114         return KEY_FOUR;
00115     case '%':
00116         return KEY_FIVE;
00117     case '^':
00118         return KEY_SIX;
00119     case '&':
00120         return KEY_SEVEN;
00121     case '*':
00122         return KEY_EIGHT;
00123     case '(':
00124         return KEY_NINE;
00125     case ')':
00126         return KEY_ZERO;
00127     case '_':
00128         return KEY_MINUS;
00129     case '+':
00130         return KEY_EQUAL;
00131     case '{':
00132         return KEY_LEFT_BRACKET;
00133     case '}':
00134         return KEY_RIGHT_BRACKET;
00135     case ';':
00136         return KEY_SEMICOLON;
00137     case '"':
00138         return KEY_APOSTROPHE;
00139     case '<':
00140         return KEY_COMMA;
00141     case '>':
00142         return KEY_PERIOD;
00143     case '?':
00144         return KEY_SLASH;
00145     }
00146 }else
00147 {
00148     if(x == "esc") return KEY_ESCAPE; if(x == "enter") return KEY_ENTER;
00149     if(x == "tab") return KEY_TAB;
00150
00151     if(x == "shift") return KEY_LEFT_SHIFT;
00152     if(x == "control") return KEY_LEFT_CONTROL;
00153     if(x == "alt") return KEY_LEFT_ALT;
00154     if(x == "super") return KEY_LEFT_SUPER;
00155
00156     if(x == "right") return KEY_RIGHT;
00157     if(x == "left") return KEY_LEFT;
00158     if(x == "down") return KEY_DOWN;
00159     if(x == "up") return KEY_UP;
00160
00161     if(x == "leftshift") return KEY_LEFT_SHIFT;
00162     if(x == "leftcontrol") return KEY_LEFT_CONTROL;
00163     if(x == "leftalt") return KEY_LEFT_ALT;
00164     if(x == "leftsuper") return KEY_LEFT_SUPER;
00165     if(x == "rightshift") return KEY_RIGHT_SHIFT;
00166     if(x == "rightcontrol") return KEY_RIGHT_CONTROL;
00167     if(x == "rightalt") return KEY_RIGHT_ALT;
00168     if(x == "rightsuper") return KEY_RIGHT_SUPER;
00169     if(x == "menu") return KEY_MENU;
00170
00171     if(x == "backspace") return KEY_BACKSPACE;
00172     if(x == "insert") return KEY_INSERT;
00173     if(x == "delete") return KEY_DELETE;
00174     if(x == "pause") return KEY_PAUSE;
00175
00176     if(x == "f1") return KEY_F1;
00177     if(x == "f2") return KEY_F2;
00178     if(x == "f3") return KEY_F3;
00179     if(x == "f4") return KEY_F4;
00180     if(x == "f5") return KEY_F5;
00181     if(x == "f6") return KEY_F6;
00182     if(x == "f7") return KEY_F7;
00183     if(x == "f8") return KEY_F8;
00184     if(x == "f9") return KEY_F9;
00185     if(x == "f10") return KEY_F10;
00186     if(x == "f11") return KEY_F11;
00187     if(x == "f12") return KEY_F12;
00188
00189     if(x == "pageup") return KEY_PAGE_UP;
00190     if(x == "pagedown") return KEY_PAGE_DOWN;
00191     if(x == "home") return KEY_HOME;
00192     if(x == "end") return KEY_END;
00193     if(x == "capslock") return KEY_CAPS_LOCK;
00194     if(x == "scrolllock") return KEY_SCROLL_LOCK;
00195     if(x == "numlock") return KEY_NUM_LOCK;
00196     if(x == "printscreen") return KEY_PRINT_SCREEN;
00197 }
00198

```



```
00199     return 0;
00200 }
```

## 8.117 keystroke.hpp

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

```
00001 #ifndef KEYSTROKE_HPP
00002 #define KEYSTROKE_HPP
00003
00004 #include <action.hpp>
00005 #include <vector>
00006 #include <string>
00007
00014 int toKey(std::string);
00015
00016 class KeyStroke
00017 {
00018 private:
00019     std::vector<int> key;
00020     std::vector< Action* > action;
00021     int id;
00022 public:
00023     KeyStroke();
00024     KeyStroke(std::vector<int>);
00025     ~KeyStroke();
00026
00027     int size();
00028     void add(unsigned char);
00029     void setAction(std::vector<Action*>);
00030     void addAction(Action*);
00031     void chooseAction(int);
00032     int getCurrent(int);
00033     void nextAction();
00034
00035     Action* react();
00036 };
00037
00038 #endif
```

## 8.118 src/utils/include/random.hpp File Reference

```
#include <random>
#include <string>
```

### Classes

- class [RandomEngine](#)

## 8.119 random.hpp

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

```
00001 #ifndef RANDOM_HPP
00002 #define RANDOM_HPP
00003
00004 #include <random>
00005 #include <string>
00006
00007 class RandomEngine
00008 {
00009 private:
00010     std::mt19937 engine;
00011 public:
00012     RandomEngine();
```

```

00013     RandomEngine(unsigned int seed);
00014     ~RandomEngine();
00015     int  randInt(int min = 0, int max = 1);
00016     double randDouble(double min = 0, double max = 1);
00017     char  randChar(char min = 0, char max = 127);
00018     std::string randString(int length, char min, char max);
00019     std::string randInt2String(int length, int min = 0, int max = 9);
00020     std::string randString(int length, bool haveDigit = true, bool haveLower = true, bool haveUpper =
        true, bool haveSpecial = true);
00021 };
00022
00023
00024 #endif

```

## 8.120 src/utils/include/vector.hpp File Reference

```

#include <const/datatype.hpp>
#include <math.h>

```

### Namespaces

- namespace [VECTOR2D](#)

### Functions

- float [VECTOR2D::getAngle](#) (fPoint v1)
- float [VECTOR2D::getAngle](#) (fPoint v1, fPoint v2)

## 8.121 vector.hpp

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

```

00001 #ifndef MY_VECTOR_SPACE_HPP
00002 #define MY_VECTOR_SPACE_HPP
00003
00004 #include <const/datatype.hpp>
00005 #include <math.h>
00006
00007 namespace VECTOR2D
00008 {
00009     float getAngle(fPoint v1);
00010     float getAngle(fPoint v1, fPoint v2);
00011 };
00012
00013 #endif

```

## 8.122 src/utils/src/constant.cpp File Reference

```

#include <const/path/assets.hpp>
#include <const/path/atb.hpp>

```

## 8.123 constant.cpp

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

```
00001 #include <const/path/assets.hpp>
00002 #include <const/path/atb.hpp>
00003
00004 const std::string PASSETS::GRAPHIC_ = "assets/graphics/";
00005 const std::string PASSETS::SOUND_ = "assets/sounds/";
00006 const std::string PASSETS::FONT_ = "assets/fonts/";
00007
00008 const std::string PATB::ATB_ = "atb/";
00009 const std::string PATB::WINDOW_ = "atb/window/";
00010 const std::string PATB::INTERFACE_ = "atb/interface/";
00011 const std::string PATB::BUTTON_ = "atb/button/";
00012 const std::string PATB::CONTAINER_ = "atb/container/";
00013 const std::string PATB::OBJECT_ = "atb/object/";
00014 const std::string PATB::MAP_ = "atb/map/";
00015 const std::string PATB::CHUNK_ = "atb/chunk/";
00016 const std::string PATB::BLOCK_ = "atb/block/";
00017 const std::string PATB::ENTITY_ = "atb/entity/";
00018
```

## 8.124 src/utils/src/countdown.cpp File Reference

```
#include <countdown.hpp>
```

## 8.125 countdown.cpp

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

```
00001 #include <countdown.hpp>
00002
00003 Countdown::CountDown(int milliseconds)
00004 {
00005     start = std::chrono::system_clock::now();
00006     finished = false;
00007     elapsed_seconds = std::chrono::milliseconds(milliseconds);
00008 }
00009
00010 Countdown::~~CountDown()
00011 {
00012 }
00013
00014 bool Countdown::isFinished()
00015 {
00016     return finished || (std::chrono::system_clock::now() - start) > elapsed_seconds;
00017 }
00018
00019 void Countdown::run()
00020 {
00021     finished = false;
00022     start = std::chrono::system_clock::now();
00023 }
00024
00025 int Countdown::get()
00026 {
00027     return elapsed_seconds.count() * 1000;
00028 }
```

## 8.126 src/utils/src/file.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <file.hpp>
```

## 8.127 file.cpp

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

```
00001 #include <iostream>
00002 #include <fstream>
00003 #include <file.hpp>
00004
00005 bool YAML_FILE::isFile(std::string path)
00006 {
00007     // return true if file exists
00008
00009     std::ifstream fin(path);
00010     return fin.good();
00011 }
00012
00013 YAML::Node YAML_FILE::readFile(std::string path)
00014 {
00015     // return YAML::Node from file
00016
00017     YAML::Node node;
00018     try
00019     {
00020         node = YAML::LoadFile(path);
00021     }
00022     catch (YAML::BadFile& e)
00023     {
00024         std::cout << "Error: " << e.what() << std::endl;
00025     }
00026     return node;
00027 }
```

## 8.128 src/utils/src/keystroke.cpp File Reference

```
#include <keystroke.hpp>
#include <raylib.h>
#include <string>
#include <iostream>
```

### Functions

- int [toKey](#) (std::string x)

### 8.128.1 Function Documentation

#### 8.128.1.1 toKey()

```
int toKey (
    std::string x )
```

Definition at line 68 of file [keystroke.cpp](#).

```
00069 {
00070     if(x.size() == 1)
00071     {
00072         if(x[0] >= 'a' && x[0] <= 'z')
00073             return x[0] - 'a' + KEY_A;
00074         if(x[0] >= 'A' && x[0] <= 'Z')
00075             return x[0] - 'A' + KEY_A;
00076
00077         if(x[0] >= '0' && x[0] <= '9')
00078             return x[0] - '0' + KEY_ZERO;
00079
00080         switch (x[0]) {
00081             case ' ':
00082                 return KEY_SPACE;
```

```

00083         case '.':
00084             return KEY_PERIOD;
00085         case ',':
00086             return KEY_COMMA;
00087         case ';':
00088             return KEY_SEMICOLON;
00089         case '\\':
00090             return KEY_APOSTROPHE;
00091         case '/':
00092             return KEY_SLASH;
00093         case '\\\\':
00094             return KEY_BACKSLASH;
00095         case '-':
00096             return KEY_MINUS;
00097         case '=':
00098             return KEY_EQUAL;
00099         case '[':
00100             return KEY_LEFT_BRACKET;
00101         case ']':
00102             return KEY_RIGHT_BRACKET;
00103         case '`':
00104             return KEY_GRAVE;
00105         case '~':
00106             return KEY_GRAVE;
00107         case '!':
00108             return KEY_ONE;
00109         case '@':
00110             return KEY_ONE;
00111         case '#':
00112             return KEY_THREE;
00113         case '$':
00114             return KEY_FOUR;
00115         case '%':
00116             return KEY_FIVE;
00117         case '^':
00118             return KEY_SIX;
00119         case '&':
00120             return KEY_SEVEN;
00121         case '*':
00122             return KEY_EIGHT;
00123         case '(':
00124             return KEY_NINE;
00125         case ')':
00126             return KEY_ZERO;
00127         case '_':
00128             return KEY_MINUS;
00129         case '+':
00130             return KEY_EQUAL;
00131         case '{':
00132             return KEY_LEFT_BRACKET;
00133         case '}':
00134             return KEY_RIGHT_BRACKET;
00135         case ':':
00136             return KEY_SEMICOLON;
00137         case '"':
00138             return KEY_APOSTROPHE;
00139         case '<':
00140             return KEY_COMMA;
00141         case '>':
00142             return KEY_PERIOD;
00143         case '?':
00144             return KEY_SLASH;
00145     }
00146 }else
00147 {
00148     if(x == "esc") return KEY_ESCAPE; if(x == "enter") return KEY_ENTER;
00149     if(x == "tab") return KEY_TAB;
00150
00151     if(x == "shift") return KEY_LEFT_SHIFT;
00152     if(x == "control") return KEY_LEFT_CONTROL;
00153     if(x == "alt") return KEY_LEFT_ALT;
00154     if(x == "super") return KEY_LEFT_SUPER;
00155
00156     if(x == "right") return KEY_RIGHT;
00157     if(x == "left") return KEY_LEFT;
00158     if(x == "down") return KEY_DOWN;
00159     if(x == "up") return KEY_UP;
00160
00161     if(x == "leftshift") return KEY_LEFT_SHIFT;
00162     if(x == "leftcontrol") return KEY_LEFT_CONTROL;
00163     if(x == "leftalt") return KEY_LEFT_ALT;
00164     if(x == "leftsuper") return KEY_LEFT_SUPER;
00165     if(x == "rightshift") return KEY_RIGHT_SHIFT;
00166     if(x == "rightcontrol") return KEY_RIGHT_CONTROL;
00167     if(x == "rightalt") return KEY_RIGHT_ALT;
00168     if(x == "rightsuper") return KEY_RIGHT_SUPER;
00169     if(x == "menu") return KEY_MENU;

```

```

00170
00171     if(x == "backspace") return KEY_BACKSPACE;
00172     if(x == "insert") return KEY_INSERT;
00173     if(x == "delete") return KEY_DELETE;
00174     if(x == "pause") return KEY_PAUSE;
00175
00176     if(x == "f1") return KEY_F1;
00177     if(x == "f2") return KEY_F2;
00178     if(x == "f3") return KEY_F3;
00179     if(x == "f4") return KEY_F4;
00180     if(x == "f5") return KEY_F5;
00181     if(x == "f6") return KEY_F6;
00182     if(x == "f7") return KEY_F7;
00183     if(x == "f8") return KEY_F8;
00184     if(x == "f9") return KEY_F9;
00185     if(x == "f10") return KEY_F10;
00186     if(x == "f11") return KEY_F11;
00187     if(x == "f12") return KEY_F12;
00188
00189     if(x == "pageup") return KEY_PAGE_UP;
00190     if(x == "pagedown") return KEY_PAGE_DOWN;
00191     if(x == "home") return KEY_HOME;
00192     if(x == "end") return KEY_END;
00193     if(x == "capslock") return KEY_CAPS_LOCK;
00194     if(x == "scrolllock") return KEY_SCROLL_LOCK;
00195     if(x == "numlock") return KEY_NUM_LOCK;
00196     if(x == "printscreen") return KEY_PRINT_SCREEN;
00197 }
00198
00199 return 0;
00200 }

```

## 8.129 keystroke.cpp

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

```

00001 #include <keystroke.hpp>
00002 #include <raylib.h>
00003 #include <string>
00004 #include <iostream>
00005
00006 KeyStroke::KeyStroke()
00007 {
00008     id = 0;
00009 }
00010
00011 KeyStroke::KeyStroke(std::vector<int> k)
00012 {
00013     key = k;
00014     id = 0;
00015 }
00016
00017 KeyStroke::~KeyStroke()
00018 {
00019     for(auto &a : action)
00020     {
00021         delete a;
00022     }
00023 }
00024
00025 int KeyStroke::size()
00026 {
00027     return key.size();
00028 }
00029
00030 void KeyStroke::add(unsigned char k)
00031 {
00032     key.push_back(k);
00033 }
00034 void KeyStroke::setAction(std::vector<Action*> a)
00035 {
00036     action = a;
00037 }
00038
00039 void KeyStroke::addAction(Action* a)
00040 {
00041     action.push_back(a);
00042 }
00043
00044 Action* KeyStroke::react()
00045 {
00046     for(auto k : key)

```

```

00047     {
00048         if(!IsKeyDown(k)) return nullptr;
00049     }
00050     return action[id]->clone();
00051 }
00052 void KeyStroke::chooseAction(int i)
00053 {
00054     id = i;
00055 }
00056
00057 int KeyStroke::getCurrent(int i)
00058 {
00059     return id;
00060 }
00061
00062 void KeyStroke::nextAction()
00063 {
00064     id = (id + 1) % action.size();
00065 }
00066
00067
00068 int toKey(std::string x)
00069 {
00070     if(x.size() == 1)
00071     {
00072         if(x[0] >= 'a' && x[0] <= 'z')
00073             return x[0] - 'a' + KEY_A;
00074         if(x[0] >= 'A' && x[0] <= 'Z')
00075             return x[0] - 'A' + KEY_A;
00076
00077         if(x[0] >= '0' && x[0] <= '9')
00078             return x[0] - '0' + KEY_ZERO;
00079
00080         switch (x[0]) {
00081             case ' ':
00082                 return KEY_SPACE;
00083             case '.':
00084                 return KEY_PERIOD;
00085             case ',':
00086                 return KEY_COMMA;
00087             case ';':
00088                 return KEY_SEMICOLON;
00089             case '"':
00090                 return KEY_APOSTROPHE;
00091             case '/':
00092                 return KEY_SLASH;
00093             case '\\':
00094                 return KEY_BACKSLASH;
00095             case '-':
00096                 return KEY_MINUS;
00097             case '=':
00098                 return KEY_EQUAL;
00099             case '[':
00100                 return KEY_LEFT_BRACKET;
00101             case ']':
00102                 return KEY_RIGHT_BRACKET;
00103             case '`':
00104                 return KEY_GRAVE;
00105             case '~':
00106                 return KEY_GRAVE;
00107             case '!':
00108                 return KEY_ONE;
00109             case '@':
00110                 return KEY_ONE;
00111             case '#':
00112                 return KEY_THREE;
00113             case '$':
00114                 return KEY_FOUR;
00115             case '%':
00116                 return KEY_FIVE;
00117             case '^':
00118                 return KEY_SIX;
00119             case '&':
00120                 return KEY_SEVEN;
00121             case '*':
00122                 return KEY_EIGHT;
00123             case '(':
00124                 return KEY_NINE;
00125             case ')':
00126                 return KEY_ZERO;
00127             case '_':
00128                 return KEY_MINUS;
00129             case '+':
00130                 return KEY_EQUAL;
00131             case '{':
00132                 return KEY_LEFT_BRACKET;
00133             case '}':

```

```

00134         return KEY_RIGHT_BRACKET;
00135     case '':
00136         return KEY_SEMICOLON;
00137     case '"':
00138         return KEY_APOSTROPHE;
00139     case '<':
00140         return KEY_COMMA;
00141     case '>':
00142         return KEY_PERIOD;
00143     case '?':
00144         return KEY_SLASH;
00145     }
00146 }else
00147 {
00148     if(x == "esc") return KEY_ESCAPE; if(x == "enter") return KEY_ENTER;
00149     if(x == "tab") return KEY_TAB;
00150
00151     if(x == "shift") return KEY_LEFT_SHIFT;
00152     if(x == "control") return KEY_LEFT_CONTROL;
00153     if(x == "alt") return KEY_LEFT_ALT;
00154     if(x == "super") return KEY_LEFT_SUPER;
00155
00156     if(x == "right") return KEY_RIGHT;
00157     if(x == "left") return KEY_LEFT;
00158     if(x == "down") return KEY_DOWN;
00159     if(x == "up") return KEY_UP;
00160
00161     if(x == "leftshift") return KEY_LEFT_SHIFT;
00162     if(x == "leftcontrol") return KEY_LEFT_CONTROL;
00163     if(x == "leftalt") return KEY_LEFT_ALT;
00164     if(x == "leftsuper") return KEY_LEFT_SUPER;
00165     if(x == "rightshift") return KEY_RIGHT_SHIFT;
00166     if(x == "rightcontrol") return KEY_RIGHT_CONTROL;
00167     if(x == "rightalt") return KEY_RIGHT_ALT;
00168     if(x == "rightsuper") return KEY_RIGHT_SUPER;
00169     if(x == "menu") return KEY_MENU;
00170
00171     if(x == "backspace") return KEY_BACKSPACE;
00172     if(x == "insert") return KEY_INSERT;
00173     if(x == "delete") return KEY_DELETE;
00174     if(x == "pause") return KEY_PAUSE;
00175
00176     if(x == "f1") return KEY_F1;
00177     if(x == "f2") return KEY_F2;
00178     if(x == "f3") return KEY_F3;
00179     if(x == "f4") return KEY_F4;
00180     if(x == "f5") return KEY_F5;
00181     if(x == "f6") return KEY_F6;
00182     if(x == "f7") return KEY_F7;
00183     if(x == "f8") return KEY_F8;
00184     if(x == "f9") return KEY_F9;
00185     if(x == "f10") return KEY_F10;
00186     if(x == "f11") return KEY_F11;
00187     if(x == "f12") return KEY_F12;
00188
00189     if(x == "pageup") return KEY_PAGE_UP;
00190     if(x == "pagedown") return KEY_PAGE_DOWN;
00191     if(x == "home") return KEY_HOME;
00192     if(x == "end") return KEY_END;
00193     if(x == "capslock") return KEY_CAPS_LOCK;
00194     if(x == "scrolllock") return KEY_SCROLL_LOCK;
00195     if(x == "numlock") return KEY_NUM_LOCK;
00196     if(x == "printscreen") return KEY_PRINT_SCREEN;
00197 }
00198
00199 return 0;
00200 }

```

## 8.130 src/utls/src/random.cpp File Reference

```

#include <random.hpp>
#include <chrono>

```

### 8.131 random.cpp

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



```

00001 #include <random.hpp>
00002
00003 #include <chrono>
00004
00005
00006 RandomEngine::RandomEngine()
00007 {
00008     unsigned int seed = std::chrono::system_clock::now().time_since_epoch().count();
00009     engine.seed(seed);
00010 }
00011
00012 RandomEngine::RandomEngine(unsigned int seed)
00013 {
00014     engine.seed(seed);
00015 }
00016
00017 RandomEngine::~~RandomEngine()
00018 {
00019 }
00020
00021 int RandomEngine::randInt(int min, int max)
00022 {
00023     std::uniform_int_distribution<int> distribution(min, max);
00024     return distribution(engine);
00025 }
00026
00027 double RandomEngine::randDouble(double min, double max)
00028 {
00029     std::uniform_real_distribution<double> distribution(min, max);
00030     return distribution(engine);
00031 }
00032
00033 char RandomEngine::randChar(char min, char max)
00034 {
00035     std::uniform_int_distribution<int> distribution(min, max);
00036     return distribution(engine);
00037 }
00038
00039 std::string RandomEngine::randString(int length, char min, char max)
00040 {
00041     std::string str;
00042     for (int i = 0; i < length; i++)
00043     {
00044         str += randChar(min, max);
00045     }
00046     return str;
00047 }
00048
00049 std::string RandomEngine::randInt2String(int length, int min, int max)
00050 {
00051     std::string str;
00052     for (int i = 0; i < length; i++)
00053     {
00054         str += std::to_string(randInt(min, max));
00055     }
00056     return str;
00057 }
00058
00059 std::string RandomEngine::randString(int length, bool haveDigit, bool haveLower, bool haveUpper, bool
haveSpecial)
00060 {
00061     std::string str;
00062     std::string digit = "0123456789";
00063     std::string lower = "abcdefghijklmnopqrstuvwxyz";
00064     std::string upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
00065     std::string special = "!@#%^&*()_+{|<:>?~`-=[]\|\\';,./\"";
00066     std::string all = digit + lower + upper + special;
00067     if (haveDigit)
00068     {
00069         str += digit;
00070     }
00071     if (haveLower)
00072     {
00073         str += lower;
00074     }
00075     if (haveUpper)
00076     {
00077         str += upper;
00078     }
00079     if (haveSpecial)
00080     {
00081         str += special;
00082     }
00083     if (str.empty())
00084     {
00085         str = all;
00086     }

```

```

00087     std::string result;
00088     for (int i = 0; i < length; i++)
00089     {
00090         result += str[randInt(0, str.size() - 1)];
00091     }
00092     return result;
00093 }

```

## 8.132 src/utills/src/vector.cpp File Reference

```
#include <vector.hpp>
```

### Functions

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

### 8.132.1 Function Documentation

#### 8.132.1.1 [sqr\(\)](#)

```
float sqr (
    float x )
```

Definition at line 11 of file [vector.cpp](#).

```

00012 {
00013     return x * x;
00014 }

```

## 8.133 vector.cpp

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

```

00001 #include <vector.hpp>
00002
00003 using namespace VECTOR2D;
00004
00005 float VECTOR2D::getAngle(fPoint v1)
00006 {
00007     // arctan(y / x)
00008     return atan2(v1[1], v1[0]);
00009 }
00010
00011 float sqr(float x)
00012 {
00013     return x * x;
00014 }
00015
00016 float VECTOR2D::getAngle(fPoint v1, fPoint v2)
00017 {
00018     // angle between 2 vector
00019     // v1 * v2 = |v1| * |v2| * cos(angle)
00020
00021     float dot = v1[0] * v2[0] + v1[1] * v2[1];
00022     float abs1 = sqrt(sqr(v1[0]) + sqr(v1[1]));
00023     float abs2 = sqrt(sqr(v2[0]) + sqr(v2[1]));
00024     return acos(dot / (abs1 * abs2));
00025 }

```

## 8.134 src/visual/include/visual.hpp File Reference

```
#include <memory>
#include <raylib.h>
#include <yaml-cpp/yaml.h>
#include <frame.hpp>
```

### Classes

- class [Visual](#)  
*images displayed on screen*

### Typedefs

- using [Sprite](#) = std::vector<[Visual](#)\*>

### Functions

- void [deleteSprite](#) ([Sprite](#) \*&)
- void [deleteSprites](#) (std::vector< [Sprite](#) > \*&)

### 8.134.1 Typedef Documentation

#### 8.134.1.1 Sprite

```
using Sprite = std::vector<Visual*>
```

Definition at line 36 of file [visual.hpp](#).

### 8.134.2 Function Documentation

#### 8.134.2.1 deleteSprite()

```
void deleteSprite (  
    Sprite *& )
```

#### 8.134.2.2 deleteSprites()

```
void deleteSprites (  
    std::vector< Sprite > *& sprites )
```

Definition at line 24 of file [destructor.cpp](#).

```
00025 {  
00026     for(Sprite & sprite : *sprites)  
00027     {  
00028         deleteSprite(sprite);  
00029     }  
00030     delete sprites;  
00031 }
```

## 8.135 visual.hpp

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

```

00001 #ifndef VISUAL
00002 #define VISUAL
00003
00004 #include <memory>
00005 #include <raylib.h>
00006 #include <yaml-cpp/yaml.h>
00007
00008 #include <frame.hpp>
00009
00016 class Visual : public Frame
00017 {
00018 private:
00019     std::shared_ptr<Texture2D> m_texture;
00020     bool resizeable;
00021     static void deleteTexture2D(Texture2D*&);
00022 protected:
00023     void fitFrame();
00024     void updateFrame(bool recursive = false) override;
00025 public:
00026     Visual(Texture2D*, Frame*, Rectangle);
00027     Visual(Visual*);
00028     Visual(Visual*, Rectangle);
00029     Visual(Visual*, Frame*, Rectangle);
00030     ~Visual();
00031
00032     void resize(fPoint);
00033
00034     void draw();
00035 };
00036 using Sprite = std::vector<Visual*>;
00037 void deleteSprite(Sprite*&);
00038 void deleteSprites(std::vector<Sprite*>&);
00039 #endif // VISUAL

```

## 8.136 src/window/include/window.hpp File Reference

```

#include <iostream>
#include <thread>
#include <chrono>
#include <vector>
#include <queue>
#include <stack>
#include <raylib.h>
#include <visual.hpp>
#include <action.hpp>
#include <container.hpp>
#include <object.hpp>
#include <interface.hpp>
#include <button.hpp>

```

### Classes

- class [Window](#)
- class [CloseAction](#)  
*manages the closing of the application*
- class [resizeAction](#)  
*manages the resizing of the window*

## 8.137 window.hpp

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

```

00001 #ifndef WINDOW_H
00002 #define WINDOW_H
00003
00004 #include <iostream>
00005 #include <thread>
00006 #include <chrono>
00007 #include <vector>
00008 #include <queue>
00009 #include <stack>
00010
00011 #include <raylib.h>
00012
00013 #include <visual.hpp>
00014 #include <action.hpp>
00015 #include <container.hpp>
00016 #include <object.hpp>
00017 #include <interface.hpp>
00018 #include <button.hpp>
00019
00020
00021 class Window
00022 {
00023 private:
00024     class InterfacePool
00025     {
00026     private:
00027         std::stack<Interface*> inf;
00028         std::map<std::string, Interface*> storage;
00029         void clearStack();
00030     public:
00031         InterfacePool();
00032         ~InterfacePool();
00033         void load(Interface*);
00034         void unload(Interface*);
00035         void clear();
00036         Interface* getInterface(std::string);
00037
00038         void push(std::string);
00039         std::string pop();
00040         Interface* top();
00041
00042         void draw();
00043         Action* react();
00044         Action* getRuntimeEvent();
00045
00046     };
00047     class ActionPool
00048     {
00049     private:
00050         std::queue<Action*> pool;
00051         std::mutex mtx;
00052     public:
00053         ActionPool() = default;
00054         ~ActionPool();
00055         void push(Action* act);
00056         void push(PacketAction* act);
00057         Action* front();
00058         Action* pop();
00059         bool empty();
00060     };
00061     class WinContent
00062     {
00063     private:
00064         bool status;
00065         std::chrono::time_point<std::chrono::steady_clock> input_clock;
00066         std::chrono::time_point<std::chrono::steady_clock> runtime_clock;
00067
00068         std::mutex status_mtx;
00069         std::mutex input_mtx;
00070         std::mutex runtime_mtx;
00071     public:
00072         std::chrono::duration<double> input_delay;
00073         std::chrono::duration<double> runtime_delay;
00074         float width;
00075         float height;
00076         Color background;
00077         std::string title;
00078
00079         std::vector<std::thread> thread_pool;
00080
00081         ~WinContent();
00082

```

```

00083         void setStatus(bool);
00084         bool getStatus();
00085
00086         void setInputClock2Now();
00087         void setRuntimeClock2Now();
00088
00089         bool isInputDelayOver();
00090         bool isRuntimeDelayOver();
00091
00092     };
00093     class UI
00094     {
00095     private:
00096         Frame* root_frame;
00097         InterfacePool* interface;
00098
00099         std::mutex mtx;
00100         int reader;
00101         int writer;
00102         bool noRead;
00103         bool noWrite;
00104     protected:
00105         bool isReadable();
00106         bool isWritable();
00107
00108         void reading();
00109         bool tryReading();
00110         void endReading();
00111
00112         void writing();
00113         bool tryWriting();
00114         void endWriting();
00115
00116         void DenyRead();
00117         void AllowRead();
00118
00119         void DenyWrite();
00120         void AllowWrite();
00121
00122     public:
00123         UI();
00124         ~UI();
00125         void draw();
00126         Action* react();
00127         Action* getRuntimeEvent();
00128
00129         void setRootFrame(Frame*);
00130         Frame* getRootFrame();
00131         void resize(float, float);
00132
00133         void setInterfacePool(InterfacePool*);
00134
00135         void load(Interface*);
00136         void unload(Interface*);
00137         Interface* getInterface(std::string);
00138
00139         void push(std::string);
00140         std::string pop();
00141         Interface* top();
00142     };
00143
00144     friend class CloseAction;
00145     friend class resizeAction;
00146
00147     WinContent Wcontent;
00148     UI UI;
00149     ActionPool immediate_user_pool, immediate_pool, request_pool, system_pool;
00150
00151     protected:
00152         void draw();
00153         void systemEvent();
00154         void getUserEvent();
00155         void getRuntimeEvent();
00156         void sound_effect();
00157         void immediateActing();
00158         void userActing();
00159         void requestActing();
00160         void systemActing();
00161
00162         void initRaylib(YAML::Node node);
00163         void loadInterface(YAML::Node node);
00164         void loadGame(YAML::Node node);
00165     public:
00166         Window();
00167         Window(std::string path);
00168         ~Window();
00169

```

```

00170     bool isRun();
00171     bool isClose();
00172     void run();
00173 };
00174
00181 class CloseAction : public Action
00182 {
00183 private:
00184     Window * win;
00185 public:
00186     CloseAction(Window* win);
00187     ~CloseAction() = default;
00188     void execute();
00189 };
00190
00197 class resizeAction : public Action
00198 {
00199 private:
00200     float w, h;
00201     Window* win;
00202 public:
00203     resizeAction(Window* window, float w, float h);
00204     ~resizeAction() = default;
00205     void execute();
00206 };
00207 #endif

```

## 8.138 src/window/src/acting.cpp File Reference

```

#include <const/request.hpp>
#include <window.hpp>

```

## 8.139 acting.cpp

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

```

00001 #include <const/request.hpp>
00002 #include <window.hpp>
00003
00004 void Window::ActionPool::push(Action* action)
00005 {
00006     std::lock_guard<std::mutex> lock(mtx);
00007     pool.push(action);
00008 }
00009
00010 void Window::ActionPool::push(PacketAction* action)
00011 {
00012     std::lock_guard<std::mutex> lock(mtx);
00013     std::vector<Action*> unpacked = action->unpack();
00014     delete action;
00015
00016     for(Action* a : unpacked)
00017     {
00018         pool.push(a);
00019     }
00020 }
00021
00022 Window::ActionPool::~ActionPool()
00023 {
00024     while(pop() != nullptr);
00025 }
00026
00027 Action* Window::ActionPool::front()
00028 {
00029     std::lock_guard<std::mutex> lock(mtx);
00030     return pool.front();
00031 }
00032
00033 Action* Window::ActionPool::pop()
00034 {
00035     std::lock_guard<std::mutex> lock(mtx);
00036     if(pool.empty()) return nullptr;
00037     Action* action = pool.front();

```

```

00039     pool.pop();
00040     return action;
00041 }
00042
00043 bool Window::ActionPool::empty()
00044 {
00045     std::lock_guard<std::mutex> lock(mtx);
00046     return pool.empty();
00047 }
00048
00049
00050 void Window::immediateActing()
00051 {
00052     while(isRun())
00053     {
00054         Action* action = immediate_pool.pop();
00055         if(action == nullptr) continue;
00056         if(!isRun()) break;
00057         action->execute();
00058         delete action;
00059     }
00060 }
00061
00062 void Window::userActing()
00063 {
00064     while(isRun())
00065     {
00066         Action* action = immediate_user_pool.pop();
00067         if(action == nullptr) continue;
00068         if(!isRun()) break;
00069         action->execute();
00070         delete action;
00071     }
00072 }
00073
00074 void Window::systemActing()
00075 {
00076     {
00077         Action* action = system_pool.pop();
00078         if(action == nullptr) return;
00079         if(!isRun()) return;
00080         action->execute();
00081         delete action;
00082     }
00083 }
00084
00085 void Window::requestActing()
00086 {
00087     while(isRun())
00088     {
00089         Action* action = request_pool.pop();
00090         if(action == nullptr) continue;
00091         if(!isRun()) break;
00092         switch(action->isRequest())
00093         {
00094             case (REQUEST::ID::NONE):
00095                 break;
00096             case (REQUEST::ID::INVALID):
00097                 break;
00098             case (REQUEST::ID::CHANGE_INF):
00099                 {
00100                     std::string id = action->getArgs().getInterfaceName();
00101                     UI.push(id);
00102                     break;
00103                 }
00104             default:
00105                 break;
00106         };
00107         delete action;
00108     }
00109 }
00110 }

```

## 8.140 src/window/src/action/close.cpp File Reference

```
#include <window.hpp>
```



## 8.141 close.cpp

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

```
00001 #include <window.hpp>
00002
00003 CloseAction::CloseAction(Window* window)
00004 {
00005     win = window;
00006 }
00007
00008 void CloseAction::execute()
00009 {
00010     win->Wcontent.setStatus(false);
00011 }
```

## 8.142 src/window/src/action/resize.cpp File Reference

```
#include <window.hpp>
```

## 8.143 resize.cpp

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

```
00001 #include <window.hpp>
00002
00003 resizeAction::resizeAction(Window* window, float x, float y)
00004 {
00005     win = window;
00006     w = x;
00007     h = y;
00008 }
00009
00010 void resizeAction::execute()
00011 {
00012     win->UI.resize(w, h);
00013 }
00014
00015
```

## 8.144 src/window/src/interface.cpp File Reference

```
#include <window.hpp>
```

## 8.145 interface.cpp

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

```
00001 #include <window.hpp>
00002
00003 void Window::InterfacePool::clearStack()
00004 {
00005     while(!inf.empty())
00006     {
00007         inf.pop();
00008     }
00009 }
00010
00011 Window::InterfacePool::InterfacePool()
00012 {
00013 }
```

```

00014
00015 Window::InterfacePool::~~InterfacePool()
00016 {
00017     clear();
00018 }
00019
00020 void Window::InterfacePool::load(Interface* i)
00021 {
00022     storage[i->getName()] = i;
00023 }
00024
00025 void Window::InterfacePool::unload(Interface* i)
00026 {
00027     if(storage.find(i->getName()) != storage.end())
00028     {
00029         storage.erase(i->getName());
00030         delete i;
00031     }
00032 }
00033
00034 void Window::InterfacePool::clear()
00035 {
00036     for(auto i : storage)
00037     {
00038         delete i.second;
00039     }
00040     storage.clear();
00041     clearStack();
00042 }
00043
00044 Interface* Window::InterfacePool::getInterface(std::string name)
00045 {
00046     if(storage.find(name) != storage.end())
00047     {
00048         return storage[name];
00049     }
00050     return nullptr;
00051 }
00052
00053 void Window::InterfacePool::push(std::string name)
00054 {
00055     if(storage.find(name) != storage.end())
00056     {
00057         inf.push(storage[name]);
00058     }
00059 }
00060
00061 std::string Window::InterfacePool::pop()
00062 {
00063     if(!inf.empty())
00064     {
00065         std::string name = inf.top()->getName();
00066         inf.pop();
00067         return name;
00068     }
00069     return "";
00070 }
00071
00072 Interface* Window::InterfacePool::top()
00073 {
00074     if(!inf.empty())
00075     {
00076         return inf.top();
00077     }
00078     return nullptr;
00079 }
00080
00081 void Window::InterfacePool::draw()
00082 {
00083     if(!inf.empty())
00084     {
00085         inf.top()->draw();
00086     }
00087 }
00088
00089 Action* Window::InterfacePool::react()
00090 {
00091     if(!inf.empty())
00092     {
00093         return inf.top()->react();
00094     }
00095     return nullptr;
00096 }
00097
00098 Action* Window::InterfacePool::getRuntimeEvent()
00099 {
00100     if(!inf.empty())

```

```

00101     {
00102         return inf.top()->getRuntimeEvent();
00103     }
00104     return nullptr;
00105 }

```

## 8.146 src/window/src/running.cpp File Reference

```

#include "raylib.h"
#include <window.hpp>

```

## 8.147 running.cpp

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

```

00001 #include "raylib.h"
00002 #include <window.hpp>
00003
00004 void Window::run() {
00005     // last_christmas = now()
00006     Wcontent.setInputClock2Now();
00007     Wcontent.setRuntimeClock2Now();
00008
00009     //Wcontent.thread_pool.push_back(std::thread(&Window::draw, this));
00010     Wcontent.thread_pool.push_back(std::thread(&Window::getUserEvent, this));
00011     Wcontent.thread_pool.push_back(std::thread(&Window::getRuntimeEvent, this));
00012     //Wcontent.thread_pool.push_back(std::thread(&Window::sound_effect, this));
00013     Wcontent.thread_pool.push_back(std::thread(&Window::userActing, this));
00014     Wcontent.thread_pool.push_back(std::thread(&Window::userActing, this));
00015     Wcontent.thread_pool.push_back(std::thread(&Window::immediateActing, this));
00016     Wcontent.thread_pool.push_back(std::thread(&Window::immediateActing, this));
00017     Wcontent.thread_pool.push_back(std::thread(&Window::immediateActing, this));
00018     Wcontent.thread_pool.push_back(std::thread(&Window::requestActing, this));
00019
00020     while (isRun())
00021     {
00022         draw();
00023         systemEvent();
00024         systemActing();
00025         //getUserEvent();
00026         //getRuntimeEvent();
00027         //sound_effect();
00028         //userActing();
00029         //immediateActing();
00030         std::this_thread::sleep_for(std::chrono::milliseconds(5));
00031     }
00032 }
00033
00034 void Window::draw()
00035 {
00036     {
00037         BeginDrawing();
00038         UI.draw();
00039         EndDrawing();
00040     }
00041 }
00042
00043
00044 void Window::systemEvent()
00045 {
00046     {
00047         // alt + F4 to exit
00048         if (IsKeyDown(KEY_LEFT_ALT) && IsKeyDown(KEY_F4))
00049         {
00050             system_pool.push(new CloseAction(this));
00051         }
00052         if (WindowShouldClose())
00053         {
00054             system_pool.push(new CloseAction(this));
00055         }
00056
00057         if (IsWindowResized() && !IsWindowFullscreen())
00058         {
00059             int width = GetScreenWidth();

```

```

00060         int height = GetScreenHeight();
00061         system_pool.push(new resizeAction(this, width, height));
00062     }
00063 }
00064 }
00065
00066 void Window::getUserEvent()
00067 {
00068     while(isRun())
00069     {
00070         if(!Wcontent.isInputDelayOver())
00071         {
00072             std::this_thread::sleep_for(std::chrono::milliseconds(10));
00073             continue;
00074         }
00075
00076         Action* action = UI.react();
00077         if(action != nullptr)
00078         {
00079             if(action->isPackage())
00080             {
00081                 for(auto act : action->unpack())
00082                 {
00083                     if(act->isRequest())
00084                         request_pool.push(act);
00085                     else
00086                         immediate_user_pool.push(act);
00087                 }
00088             }
00089             else if(!action->isRequest())
00090                 immediate_user_pool.push(action);
00091         }
00092
00093         Wcontent.setInputClock2Now();
00094     }
00095 }
00096
00097 void Window::getRuntimeEvent()
00098 {
00099     while(isRun())
00100     {
00101
00102         if(!Wcontent.isRuntimeDelayOver())
00103         {
00104             std::this_thread::sleep_for(std::chrono::milliseconds(10));
00105             continue;
00106         }
00107
00108         Action* action = UI.getRuntimeEvent();
00109         if(action != nullptr)
00110         {
00111             if(action->isPackage())
00112             {
00113                 for(auto act : action->unpack())
00114                 {
00115                     if(act->isRequest())
00116                         request_pool.push(act);
00117                     else
00118                         immediate_user_pool.push(act);
00119                 }
00120             }
00121             else if(!action->isRequest())
00122                 immediate_pool.push(action);
00123
00124             Wcontent.setRuntimeClock2Now();
00125         }
00126     }
00127
00128 void Window::sound_effect()
00129 {
00130     // do nothing
00131 }
00132
00133 bool Window::isRun()
00134 {
00135     return Wcontent.getStatus();
00136 }
00137
00138 bool Window::isClose()
00139 {
00140     return !Wcontent.getStatus();
00141 }

```

## 8.148 src/window/src/UI.cpp File Reference

```
#include "action.hpp"
#include <window.hpp>
#include <mutex>
```

## 8.149 UI.cpp

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

```
00001 #include "action.hpp"
00002 #include <window.hpp>
00003 #include <mutex>
00004
00005 Window::UI::UI()
00006 {
00007     root_frame = nullptr;
00008     interface = nullptr;
00009 }
00010
00011 Window::UI::~~UI()
00012 {
00013     if(root_frame != nullptr) delete root_frame;
00014     if(interface != nullptr) delete interface;
00015 }
00016
00017 void Window::UI::draw()
00018 {
00019     if(!tryReading()) return ;
00020     interface->draw();
00021     endReading();
00022 }
00023
00024 Action* Window::UI::react()
00025 {
00026     if(!tryReading()) return nullptr;
00027     Action* act = interface->react();
00028     endReading();
00029
00030     return act;
00031 }
00032
00033 Action* Window::UI::getRuntimeEvent()
00034 {
00035     if(!tryReading()) return nullptr;
00036     Action* act = interface->getRuntimeEvent();
00037     endReading();
00038
00039     return act;
00040 }
00041
00042 void Window::UI::setRootFrame(Frame* frame)
00043 {
00044     if(root_frame != nullptr)
00045         delete root_frame;
00046     root_frame = frame;
00047 }
00048
00049 Frame* Window::UI::getRootFrame()
00050 {
00051     return root_frame;
00052 }
00053
00054 void Window::UI::resize(float width, float height)
00055 {
00056     writing();
00057     root_frame->resize(width, height);
00058     endWriting();
00059 }
00060
00061 void Window::UI::setInterfacePool(InterfacePool* inter)
00062 {
00063     if(interface != nullptr)
00064         delete interface;
00065     interface = inter;
00066 }
00067
```

```

00068 void Window::UI::load(Interface* inf)
00069 {
00070     interface->load(inf);
00071 }
00072
00073 void Window::UI::unload(Interface* inf)
00074 {
00075     interface->unload(inf);
00076 }
00077
00078 Interface* Window::UI::getInterface(std::string s)
00079 {
00080     reading();
00081     Interface* f = interface->getInterface(s);
00082     endReading();
00083     return f;
00084 }
00085
00086
00087 void Window::UI::push(std::string s)
00088 {
00089     writing();
00090     interface->push(s);
00091     endWriting();
00092 }
00093
00094 std::string Window::UI::pop()
00095 {
00096     writing();
00097     std::string f = interface->pop();
00098     endWriting();
00099     return f;
00100 }
00101
00102 Interface* Window::UI::top()
00103 {
00104     reading();
00105     Interface* f = interface->top();
00106     endReading();
00107     return f;
00108 }
00109
00110 bool Window::UI::isReadable()
00111 {
00112     std::lock_guard<std::mutex> lock(mtx);
00113     return !noRead;
00114 }
00115
00116 bool Window::UI::isWritable()
00117 {
00118     std::lock_guard<std::mutex> lock(mtx);
00119     return !noWrite;
00120 }
00121
00122 void Window::UI::reading()
00123 {
00124     do
00125     {
00126         std::this_thread::sleep_for(std::chrono::milliseconds(5));
00127         std::lock_guard<std::mutex> lock(mtx);
00128         if(noRead) return ;
00129         reader++;
00130         return ;
00131     }while(true);
00132 }
00133
00134 bool Window::UI::tryReading()
00135 {
00136     std::lock_guard<std::mutex> lock(mtx);
00137     if(noRead) return false;
00138     reader++;
00139     return true;
00140 }
00141
00142 void Window::UI::endReading()
00143 {
00144     std::lock_guard<std::mutex> lock(mtx);
00145     reader--;
00146 }
00147
00148 void Window::UI::writing()
00149 {
00150     do
00151     {
00152         std::this_thread::sleep_for(std::chrono::milliseconds(5));
00153         std::lock_guard<std::mutex> lock(mtx);

```

```

00155         if(noWrite) return ;
00156         writer++;
00157         noRead = true;
00158         return;
00159     }while(true);
00160 }
00161
00162 bool Window::UI::tryWriting()
00163 {
00164     std::lock_guard<std::mutex> lock(mtx);
00165     if(noWrite) return false;
00166     writer++;
00167     noRead = true;
00168     return true;
00169 }
00170
00171 void Window::UI::endWriting()
00172 {
00173     std::lock_guard<std::mutex> lock(mtx);
00174     writer--;
00175     if(writer == 0) noRead = false;
00176 }
00177
00178 void Window::UI::DenyRead()
00179 {
00180     std::lock_guard<std::mutex> lock(mtx);
00181     noRead = true;
00182 }
00183
00184 void Window::UI::AllowRead()
00185 {
00186     std::lock_guard<std::mutex> lock(mtx);
00187     noRead = false;
00188 }
00189
00190 void Window::UI::DenyWrite()
00191 {
00192     std::lock_guard<std::mutex> lock(mtx);
00193     noWrite = true;
00194 }
00195
00196 void Window::UI::AllowWrite()
00197 {
00198     std::lock_guard<std::mutex> lock(mtx);
00199     noWrite = false;
00200 }
00201
00202

```

## 8.150 src/window/src/wincontent.cpp File Reference

```
#include <window.hpp>
```

## 8.151 wincontent.cpp

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

```

00001 #include <window.hpp>
00002
00003 Window::WinContent::~WinContent()
00004 {
00005     for(std::thread &i : thread_pool)
00006     {
00007         if(i.joinable())
00008             i.join();
00009     }
00010 }
00011
00012
00013 void Window::WinContent::setStatus(bool b)
00014 {
00015     std::lock_guard<std::mutex> lock(status_mtx);
00016     status = b;
00017 }
00018

```

```
00019 bool Window::WinContent::getStatus()
00020 {
00021     std::lock_guard<std::mutex> lock(status_mtx);
00022     return status;
00023 }
00024
00025 void Window::WinContent::setInputClock2Now()
00026 {
00027     std::lock_guard<std::mutex> lock(input_mtx);
00028     input_clock = std::chrono::steady_clock::now();
00029 }
00030
00031 void Window::WinContent::setRuntimeClock2Now()
00032 {
00033     std::lock_guard<std::mutex> lock(runtime_mtx);
00034     runtime_clock = std::chrono::steady_clock::time_point();
00035 }
00036
00037 bool Window::WinContent::isInputDelayOver()
00038 {
00039     std::lock_guard<std::mutex> lock(input_mtx);
00040     return std::chrono::steady_clock::now() - input_clock > input_delay;
00041 }
00042
00043 bool Window::WinContent::isRuntimeDelayOver()
00044 {
00045     std::lock_guard<std::mutex> lock(runtime_mtx);
00046     return std::chrono::steady_clock::now() - runtime_clock > runtime_delay;
00047 }
00048
```



