HÁZI FELADAT

Programozás alapjai 2.

Végleges

Galgóczy Gábor
QS2WJR

2019. március 22.

TARTALOM

1. Feladat

Saját ötleten alapuló ügyességi játék elkészítése.

Készítsen „Endless Runner” típusú játékot, ahol a játékpályán maradás a cél.
A pálya véletlenszerűen generálódik a játékos előtt, akinek feladata a jelenlegi pályaelemről egy új pályaelemre átugrani. Amennyiben ez nem sikerül, az a játék végét jelenti.

2. Feladatspecifikáció

A feladatot menürendszer segítségével oldom meg, hogy ne kelljen a programot minden egyes játék után újraindítani. Ezenkívül felkínálok egy külön menüt a játék beállításaira.

Játék menete:
A játékos minden alkalommal a kezdő platformról fix sebességgel kezdi meg útját.
Az újonnan generált platformelemek időben megjelennek a játékos előtt, hogy a felhasználó erre reagálni tudjon, és döntést hozhasson.

Játék közben érvényes játékos műveletek a platformról ugrás, és a sávváltás.
Ugrás közben a játékos elugrik az egyik platformról, és halhatatlanná válik, amíg az ugrás tart. Az ugrás befejeződik, amikor a játékos talajt (platformot) érne. Amennyiben ez nem sikerül, a játék véget ér.

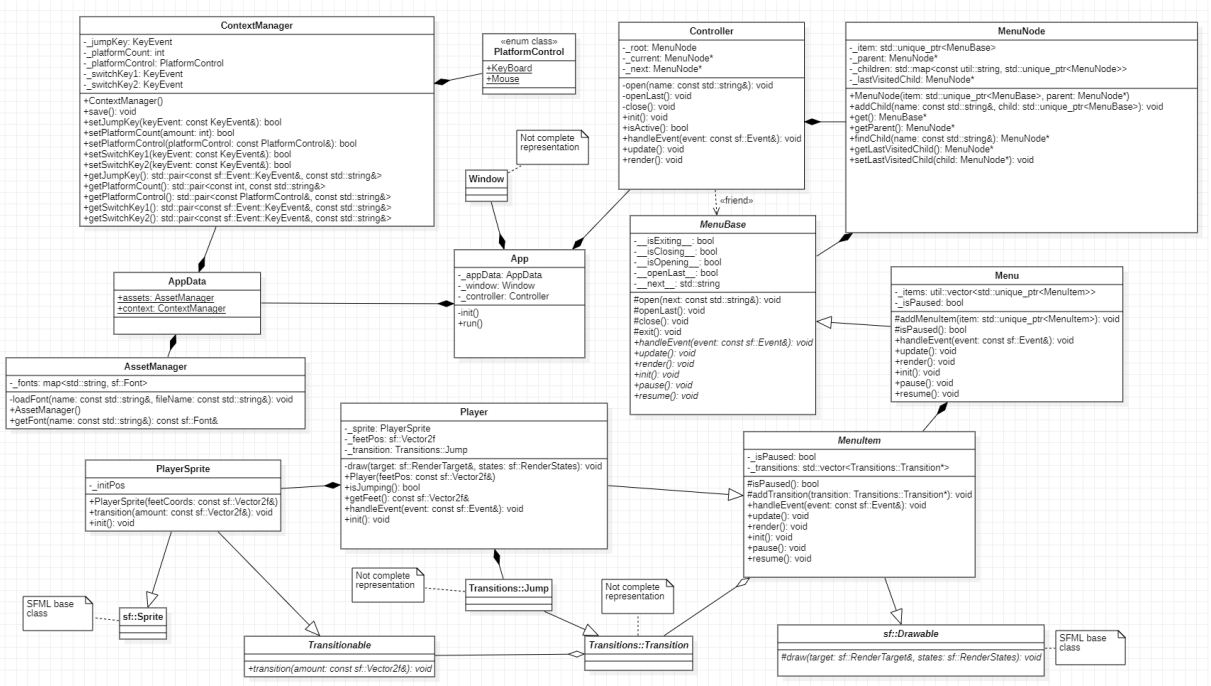
Maguk a platformok véletlenszerűen sávokba rendeződnek generáláskor. A játékos a játék bármely pontján megváltoztathatja helyét a sávok között.

Felhasználói segédlet:

A program indításakor a töltőképernyő után a főmenü jelenik meg. Innen lehetőség adódik a beállítások menübe való átlépésre, és a játék indítására. Mindkét esetben lehetséges a főmenübe való visszalépés.
A beállítások menüben grafikus beviteli mező segítségével beállítható, hogy játék közben milyen gomb lenyomására történjen ugrás (alapértelmezett: „Space”). Sávváltáshoz pedig a jobbra/balra nyilak lenyomása szükséges. (Ilyenkor egy sávváltás történik a megfelelő irányba, a sávok lineárisan helyezkednek el egymás mellett.) A játékos folyamatosan egy irányban halad sávval párhuzamosan, ezt a játék során nem lehet megváltoztatni! Sávváltáskor a sáv váltódik alatta, mintha ő lépne jobbra vagy balra. Ha nem ugrás közben váltunk sávot, és az új sávon nincsen platform, akkor a játék véget ér.

A menük közti navigáció grafikus gombok segítségével kezelhető, amik egérkattintásra váltanak menüt. Az alkalmazás grafikus része végig egérrel navigálható.

3. Objektum Terv



4. Algoritmus Terv

virtual MenuItem::update(): calls *update*() on each *transition*
virtual MenuItem::init(): calls *init*() on each *transition*
virtual MenuItem::resume(): calls *resume*() on each *transition* and sets *isPaused* to false
virtual MenuItem::pause(): calls *pause*() on each *transition* and sets *isPaused* to true

virtual Menu::update(): calls *update*() on each *item*
virtual Menu::init(): calls *init*() on each *item*
virtual Menu::resume(): calls *resume*() on each *item* and sets *isPaused* to false
virtual Menu::pause(): calls *pause*() on each *item* and sets *isPaused* to true

MenuBase::open(next): sets *isOpening* to true and *next* to next
MenuBase::openLast(): sets *isOpening* to true and *openLast* to true
MenuBase::close(): sets *isClosing* to true
MenuBase::exit(): sets *isExiting* to true

MenuNode::get(): returns a pointer to *item*
MenuNode::findChild(const std::string& name): returns a pointer to the corresponding child, returns nullptr if not found

Controller::open(const std::string& name): sets *next* to *current->findChild(name)*, sets *_current->get()->isOpening* to false
Controller::openLast(): sets *next* to *current->getLastVisitedChild()*, sets *current->get()->openLast* to false, sets *current->get()->isOpening* to false
Controller::close(): sets *next* to *current->getParent()*, sets *current->get()->isClosing* to false
Controller::init(): resets *root* (optional), builds tree from *root*
Controller::isActive():

1. returns false if *current->get()->isExiting* is true
2. calls *close*() if *current->get()->isClosing* is true
3. in the event that *current->get()->isOpening* is true:
    a. calls *openLast()* if *current->get()->openLast* is true, else *open(current->get()->next)*
    b. in the event that *next* is not a nullptr:
        i. calls *current->get()->setLastVisitedChild(next)*
        ii. calls *next->get()->init()*
4. returns true if *next* is a nullptr
5. in the event that *current* does not equal *next*:
    a. calls *current->get()->pause()*
    b. sets *current* to *next*
    c. calls *current->get()->resume()*
6. returns true

5. Megvalósítás

Mivel a feladat előző fázisai nem tartalmazták a program teljes körű leírását, így az utólagos dokumentáció nagyban egészíti ki a megoldást.

A feladat megvalósítása kisebb eltéréseket kívánt a terv fázisban leírtaktól. Ezt jól mutatja a végleges objektum terv, illetve a dokumentum végén található Doxygen-nel készített dokumentáció. Ebben megtalálhatóak a forrásfájlok leírásai is.
Az algoritmus tervhez képest többnyire a Controller és a MenuBase osztályok algoritmusaiban esett minimális változás, ami az alapvető feladat megoldása szempontjából nem változtat sokat, leginkább eszmei értéke van.

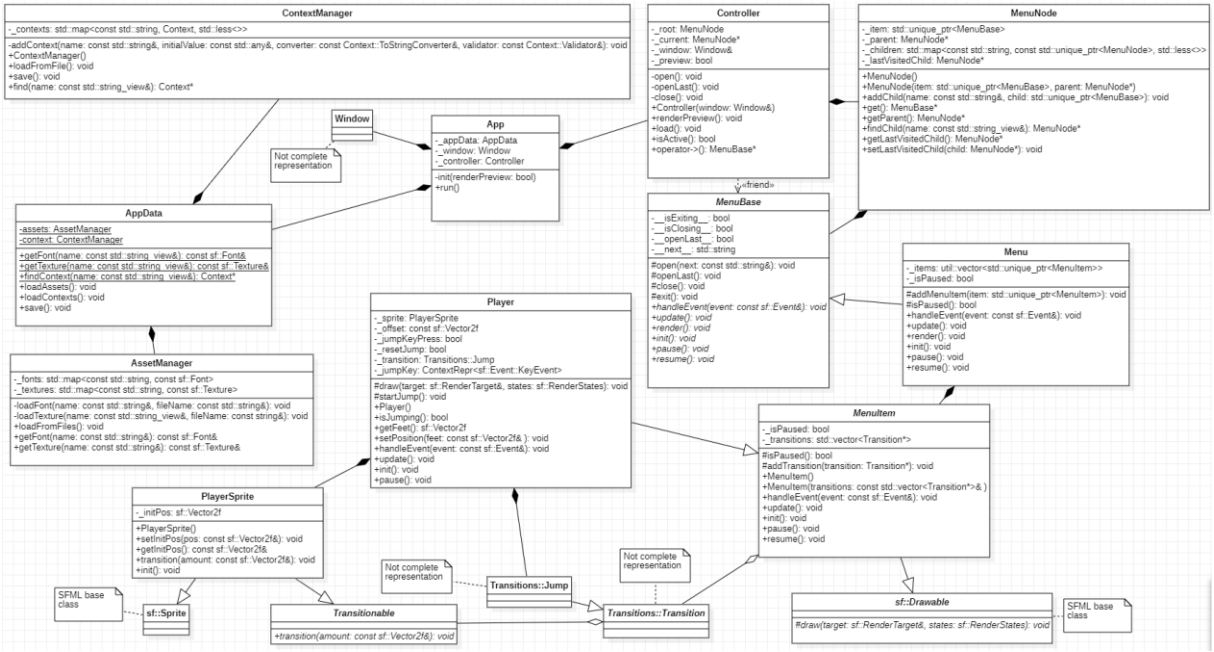
Felhasználói szempontból az eredeti specifikációhoz képest minden teljesül. Ehhez képest nagyban bővült a program által felkínált lehetőségek és beállítások száma, ezzel növelve a felhasználói élményt.

Pár billentyű kombináció, ami nem válik egyértelművé a alkalmazás használata közben:

- A főmenün belül az 1-es gomb lenyomása a játék menübe való váltást eredménye, míg a 2-es gomb lenyomása a beállítások menübe visz.
- Az utolsónak látogatott menübe való átlépéshez nyomja meg az Alt + balra nyíl kombinációt, visszalépéshez az Alt + jobbra nyilat.
- A beállítások menüpontból az Escape billentyű lenyomása a főmenübe való visszalépést eredményezi.
- Új billentyű beállítása közben Enter-rel véglegesíthető választását, vagy kapcsolhatja ki a választási folyamatot, amennyiben még nem került sor választásra.
- Játék közben az Escape lenyomása a játék megállítását eredményezi, újabb lenyomásával folytathatja a játékot.
- Miután a játék véget ért, továbbra is lehetséges az Escape-pel való perspektíva-váltás.
- Egy játék menet után Enter lenyomásával kezdhet újat, amíg nem lép vissza a főmenübe.
- A program futása alatt az Alt + F4 billentyű kombináció az alkalmazás leállását vonja maga után.

Az alkalmazás a Config mappába menti, és olvassa vissza a játék beállításait. Az itt lévő fájl átírása adatvesztést eredményezhet.

5.1. Végleges objektum terv



5.2. Tesztelés

A *test_main.cpp* fájlban lévő tesztprogramban megvalósításra kerültek a főbb osztályok unit tesztjei, ahol és amilyen tag-függvényekre ez a `gtest_lite` keretrendszerrel lehetséges volt. Ezenkívül készítettem két átfogó (integrity) tesztet és egy szimulációt is. A teszteket végreható program nem igényel felhasználói beavatkozást. Egyedül a CPORTA makró beállítása szükséges a tesztesetek fordításához az alapvető main függvény helyett.

NHF

Generated by Doxygen 1.9.3

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Namespace Documentation

5.1 math Namespace Reference

Functions for performing mathematic calculations.

Functions

- float calcDistance (const sf::Vector2f &a, const sf::Vector2f &b)

  Calculates the difference between point A and B.
- float calcAngle (const sf::Vector2f &position)

  Calculates the angle of a vector from the origin {0,0}.
- bool isBetween (float val, float smaller, float bigger)

  Checks whether a value is between two other values.
- std::vector< sf::Vector2f > getArcPoints (float angle, float spread, float radius, int maxpts)

  Calculates the points of an arc.
- template<typename T >
  T square (T num)

  Squares a number.
- float squaref (float num)

  Squares a number.
- float convertToDeg (float radian)

  Converts the given value from radians to degrees.

Variables

- const float **PI** = std::numbers::pi_v<float>

5.1.1 Detailed Description

Functions for performing mathematic calculations.

5.1.2 Function Documentation

5.1.2.1 calcAngle()

```
float math::calcAngle (
            const sf::Vector2f & position )
```

Calculates the angle of a vector from the origin {0,0}.

Parameters

position	The coordinates of the vector

Returns

The angle of the vector

5.1.2.2 calcDistance()

```
float math::calcDistance (
            const sf::Vector2f & a,
            const sf::Vector2f & b )
```

Calculates the difference between point A and B.

Parameters

a	Point A
b	Point B

Returns

The calculated distance

5.1.2.3 convertToDeg()

```
float math::convertToDeg (
            float radian )
```

Converts the given value from radians to degrees.

Parameters

radian	The value for in radians

Returns

The value in degrees

5.1.2.4 getArcPoints()

```
std::vector< sf::Vector2f > math::getArcPoints (
            float angle,
            float spread,
            float radius,
            int maxpts )
```

Calculates the points of an arc.

Parameters

angle	The middle of the arc in radians
spread	The width of the arc
radius	The radius of the arc from the origin {0,0}
maxpts	The maximum number of points constructing the arc

Returns

The points that construct the arc

5.1.2.5 isBetween()

```
bool math::isBetween (
            float val,
            float smaller,
            float bigger )
```

Checks whether a value is between two other values.

Parameters

val	The value in question
smaller	The smaller value
bigger	The bigger value

Returns

True if the value is in between

5.1.2.6 square()

```
template<typename T >
T math::square (
            T num )
```

Squares a number.

Template Parameters

T	The type of the number

Parameters

num	The number

Returns

The result

5.1.2.7 squaref()

```
float math::squaref (
            float num )  [inline]
```

Squares a number.

Parameters

num	The number

Returns

The result

5.2 theme Namespace Reference

Color themes.

Variables

- const sf::Color **Primary** = { 0, 0, 220 }
- const sf::Color **Secondary** = { 148, 0, 211 }
- const sf::Color **Tertiary** = { 0, 0, 220, 16 }
- const sf::Color **Quaternary** = { 176, 38, 255, 8 }
- const sf::Color **Gold** = { 212, 172, 43 }
- const sf::Color **Purple** = { 255, 50, 255 }
- const sf::Color **IndigoPurple** = { 75, 0, 130 }
- const sf::Color **IndigoPurpleShade** = { 75, 0, 130, 8 }
- const sf::Color **NeonYellow** = { 255, 240, 31 }

5.2.1 Detailed Description

Color themes.

5.3 Transitions Namespace Reference

Specialized transitions.

Namespaces

- namespace Bezier

  Cubic Bezier transitions.

Classes

- class EaseInOut

  Class for performing physics-like quadratic transitions (accelerate, then break)
- class Jump

  Transition class for jumping transition.

5.3.1 Detailed Description

Specialized transitions.

5.4 Transitions::Bezier Namespace Reference

Cubic Bezier transitions.

Classes

- class Ease

  Transition class for cubic Bezier easing.

5.4.1 Detailed Description

Cubic Bezier transitions.

Chapter 6

Class Documentation

6.1 App Class Reference

Class for holding the application together.

`#include <App.hpp>`

Public Member Functions

- void **run** ()
  Runs the application's processes.

6.1.1 Detailed Description

Class for holding the application together.

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

- NHF/Source/App.hpp
- NHF/Source/App.cpp

6.2 AppData Class Reference

Class for managing global application information.

`#include <AppData.hpp>`

Public Member Functions

- void **loadAssets** () const
  Loads assets from files.
- void **loadContexts** () const
  Loads contexts from file.
- void **save** () const
  Saves contexts to file.

Static Public Member Functions

- static const sf::Font & getFont (const std::string_view &name)

  Getter for font assets.
- static const sf::Texture & getTexture (const std::string_view &name)

  Getter for texture assets.
- static Context * findContext (const std::string_view &name)

  Searches for context by name.

6.2.1 Detailed Description

Class for managing global application information.

6.2.2 Member Function Documentation

6.2.2.1 findContext()

```
static Context * AppData::findContext (
            const std::string_view & name )  [inline], [static]
```

Searches for context by name.

Parameters

name	Name of the desired context

Returns

Pointer to the found context. Nullptr if not found

6.2.2.2 getFont()

```
static const sf::Font & AppData::getFont (
            const std::string_view & name )  [inline], [static]
```

Getter for font assets.

Parameters

name	Name of the font

Returns

Reference to font

6.2.2.3 getTexture()

```
static const sf::Texture & AppData::getTexture (
            const std::string_view & name )  [inline], [static]
```

Getter for texture assets.

Parameters

name	Name of the texture

Returns

Reference to texture

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

- NHF/Source/Core/AppData.hpp
- NHF/Source/Core/AppData.cpp

6.3 AssetManager Class Reference

Class for managing assets of the application.

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

Public Member Functions

- const sf::Font & getFont (const std::string_view &name)

  Getter for fonts. Returns reference to first font stored inside the class if one with the given name doesn't exist.
- const sf::Texture & getTexture (const std::string_view &name)

  Getter for textures. Returns reference to first texture stored inside the class if one with the given name doesn't exist.
- void **loadFromFiles** ()

  Reads and stores assets from the corresponding files.

6.3.1 Detailed Description

Class for managing assets of the application.

6.3.2 Member Function Documentation

6.3.2.1 getFont()

```
const sf::Font & AssetManager::getFont (
            const std::string_view & name )
```

Getter for fonts. Returns reference to first font stored inside the class if one with the given name doesn't exist.

Parameters

name	Name of the font

Returns

Reference to font

6.3.2.2 getTexture()

```
const sf::Texture & AssetManager::getTexture (
            const std::string_view & name )
```

Getter for textures. Returns reference to first texture stored inside the class if one with the given name doesn't exist.

Parameters

name	Name of the texture

Returns

Reference to texture

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

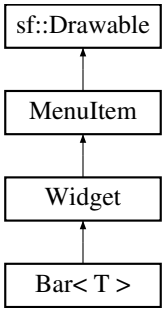
- NHF/Source/Core/AppData/AssetManager.hpp
- NHF/Source/Core/AppData/AssetManager.cpp

6.4 Bar< T > Class Template Reference

Class that acts as a bar widget. Takes advantage of the context system under the hood.

```
#include <Bar.hpp>
```

Inheritance diagram for Bar< T >:



Public Member Functions

- Bar (float width, const std::vector< T > &contents, const sf::Font &font, unsigned chararcterSize, const std↩ ::string_view &contextName)

  Constructs a new Bar.
- void setPosition (const sf::Vector2f &position) override

  Setter for the Bar's position.
- void handleEvent (const sf::Event &event) override

  Handles user input.
- void update () override

  Updates Bar.
- void init () override

  Initializes Bar.

Additional Inherited Members

6.4.1 Detailed Description

template<typename T>
class Bar< T >

Class that acts as a bar widget. Takes advantage of the context system under the hood.

Template Parameters

T	type of context data that the class manages

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Bar()

```
template<typename T >
Bar< T >::Bar (
```

```
            float width,
            const std::vector< T > & contents,
            const sf::Font & font,
            unsigned chararcterSize,
            const std::string_view & contextName )
```

Constructs a new Bar.

Parameters

width	The width of the Bar
contents	The possible variables presented to the user to chose from
font	Font style
chararcterSize	Character size of the displayed text
contextName	Name of the context that the bar is linked with

6.4.3 Member Function Documentation

6.4.3.1 handleEvent()

```
template<typename T >
void Bar< T >::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input.

Parameters

event	User input in a form of "event"

Reimplemented from MenuItem.

6.4.3.2 init()

```
template<typename T >
void Bar< T >::init  [override], [virtual]
```

Initializes Bar.

Reimplemented from MenuItem.

6.4.3.3 setPosition()

```
template<typename T >
void Bar< T >::setPosition (
            const sf::Vector2f & position )  [override], [virtual]
```

Setter for the Bar's position.

Parameters

position	New position

Reimplemented from Widget.

6.4.3.4 update()

```
template<typename T >
void Bar< T >::update  [override], [virtual]
```

Updates Bar.

Reimplemented from MenuItem.

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

- NHF/Source/GUI/Widgets/Bar.hpp

6.5 BezierEasing Class Reference

Credits for https://github.com/Trodek/Bezier-Easing ///.

```
#include <BezierEasing.hpp>
```

Public Member Functions

- BezierEasing (const sf::Vector2f &p1, const sf::Vector2f &p2)

  Creates a cubic Bezier easing for any p1 and p2 with components between 0 and 1.
- float GetEasingProgress (float t)

  Calculates progress for desired time if valid curve. -1 if invalid.

6.5.1 Detailed Description

Credits for https://github.com/Trodek/Bezier-Easing ///.

Class for calculating cubic bezier easing

6.5.2 Constructor & Destructor Documentation

6.5.2.1 BezierEasing()

```
BezierEasing::BezierEasing (
            const sf::Vector2f & p1,
            const sf::Vector2f & p2 )
```

Creates a cubic Bezier easing for any p1 and p2 with components between 0 and 1.

Parameters

p1	One point of the easing
p2	Another point of the easing

6.5.3 Member Function Documentation

6.5.3.1 GetEasingProgress()

```
float BezierEasing::GetEasingProgress (
            float t )
```

Calculates progress for desired time if valid curve. -1 if invalid.

Parameters

t	The desired time

Returns

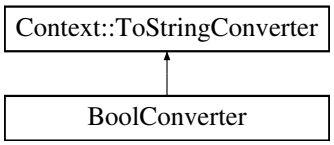
The progress of the easing from 0 to t

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

- NHF/Source/Utilities/Math/BezierEasing.hpp
- NHF/Source/Utilities/Math/BezierEasing.cpp

6.6 BoolConverter Class Reference

Inheritance diagram for BoolConverter:



Additional Inherited Members

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

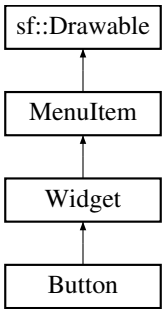
- NHF/Source/Core/AppData/ContextManager.cpp

6.7 Button Class Reference

Class that acts as a button widget.

`#include <Button.hpp>`

Inheritance diagram for Button:



Public Member Functions

- Button (const sf::String &text, const sf::Font &fontStyle, unsigned characterSize, const std::function< void()> &callback=nullptr)

  Constructs a new Button.
- void setPosition (const sf::Vector2f &position) override

  Setter for the buttons position.
- void handleEvent (const sf::Event &event) override

  Handles user input.
- void resume () override

  Resets the fill color of the button's text.

Additional Inherited Members

6.7.1 Detailed Description

Class that acts as a button widget.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Button()

```
Button::Button (
            const sf::String & text,
            const sf::Font & fontStyle,
            unsigned characterSize,
            const std::function< void()> & callback = nullptr )
```

Constructs a new Button.

Parameters

text	The text on the button
fontStyle	The font style of the text
characterSize	The character size of the text
callback	Callback function to be triggered when the button is pressed

6.7.3 Member Function Documentation

6.7.3.1 handleEvent()

```
void Button::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input.

Parameters

event	User input

Reimplemented from MenuItem.

6.7.3.2 resume()

```
void Button::resume ( )  [override], [virtual]
```

Resets the fill color of the button's text.

Reimplemented from MenuItem.

6.7.3.3 setPosition()

```
void Button::setPosition (
            const sf::Vector2f & position )  [inline], [override], [virtual]
```

Setter for the buttons position.

Parameters

position	The new position

Reimplemented from Widget.

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

- NHF/Source/GUI/Widgets/Button.hpp
- NHF/Source/GUI/Widgets/Button.cpp

6.8 Context Class Reference

Class for storing data inside ContextManager.

`#include <Context.hpp>`

Classes

- class ToStringConverter

  Functor for converting Context data to string.
- class Validator

  Functor for validating Context data.

Public Member Functions

- Context (const std::any &data, const ToStringConverter &converter, const Validator &validator)

  Constructs new context.
- template<typename T >
  T get () const

  Getter for data. Throws bad_any_cast if template parameter is not the actual type of the data store inside the class.
- std::string string () const

  Converts data stored inside class to string.
- template<typename T >
  std::string string (const T &val) const

  Uses the classes converter to try and convert the value given as parameter to string. Throws bad_any_cast if template parameter is not the actual type of the data store inside the class.
- bool set (const std::any &data)

  Validates and sets data inside class to the data given as parameter.
- bool set (std::any &&data)

  Validates and sets data inside class to the data given as parameter.
- bool validate (const std::any &potentialData) const

  Performs validation on the potential new data given as parameter.

6.8.1 Detailed Description

Class for storing data inside ContextManager.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Context()

```
Context::Context (
            const std::any & data,
            const ToStringConverter & converter,
            const Validator & validator )  [explicit]
```

Constructs new context.

Parameters

data	Data managed by class
converter	Functor for converting data to string
validator	Functor for validating new data

6.8.3 Member Function Documentation

6.8.3.1 get()

```
template<typename T >
T Context::get ( ) const  [inline]
```

Getter for data. Throws bad_any_cast if template parameter is not the actual type of the data store inside the class.

Template Parameters

T	Type of data

Returns

Data

6.8.3.2 set() [1/2]

```
bool Context::set (
            const std::any & data )
```

Validates and sets data inside class to the data given as parameter.

Parameters

data	New data

Returns

Result of validation

6.8.3.3 set() [2/2]

```
bool Context::set (
            std::any && data )
```

Validates and sets data inside class to the data given as parameter.

Parameters

data	New data

Returns

Result of validation

6.8.3.4 string() [1/2]

```
std::string Context::string ( ) const  [inline]
```

Converts data stored inside class to string.

Returns

Result of conversion

6.8.3.5 string() [2/2]

```
template<typename T >
std::string Context::string (
            const T & val ) const  [inline]
```

Uses the classes converter to try and convert the value given as parameter to string. Throws bad_any_cast if template parameter is not the actual type of the data store inside the class.

Template Parameters

T	Type of the given value

Parameters

val	Value to be converted

Returns

Result of conversion

6.8.3.6 validate()

```
bool Context::validate (
            const std::any & potentialData ) const  [inline]
```

Performs validation on the potential new data given as parameter.

Parameters

potentialData	Potential context data for validation

Returns

Result of validation

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

- NHF/Source/Core/AppData/Context.hpp
- NHF/Source/Core/AppData/Context.cpp

6.9 ContextManager Class Reference

Class for accessing and writing contexts.

`#include <ContextManager.hpp>`

Public Member Functions

- **ContextManager** ()

  Fills container with pre-defined contexts.
- void **loadFromFile** ()

  Tries reading contexts from file to class.
- void **save** ()

  Saves contexts to file.
- Context * find (const std::string_view &name)

  Searches context by name. Returns nullptr if not found.

6.9.1 Detailed Description

Class for accessing and writing contexts.

6.9.2 Member Function Documentation

6.9.2.1 find()

```
Context * ContextManager::find (
            const std::string_view & name )
```

Searches context by name. Returns nullptr if not found.

Parameters

name	Name of the searched context

Returns

Pointer to the desired context, nullptr if not found

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

- NHF/Source/Core/AppData/ContextManager.hpp
- NHF/Source/Core/AppData/ContextManager.cpp

6.10 ContextRepr< T > Class Template Reference

Class for representing a context of ContextManager. Stores a copy of data inside context that may not be up-to-date.

`#include <ContextRepr.hpp>`

Public Member Functions

- ContextRepr (Context *const context)
  Constructs new representation of a context.
- ContextRepr< T > & operator= (const T &data)
  Calls set(copy) on the context with argument given as parameter. Updates data represented locally.
- ContextRepr< T > & operator= (const T &&data)
  Calls set(move) on the context with argument given as parameter. Updates data represented locally.
- false **operator T** () const
  Default cast to of represented data to type T. No actual casting is performed. Returns data stored inside class.
- **operator std::string** () const
  Calls string function of context.
- std::string string (const T &val) const
  Equivalent to Context::string()
- bool validate (const T &potentialData) const
  Equivalent to Context::validate()
- void **update** ()
  Fetches and updates data stored inside class.

6.10.1 Detailed Description

template<typename T>
class ContextRepr< T >

Class for representing a context of ContextManager. Stores a copy of data inside context that may not be up-to-date.

Template Parameters

T	Type of data inside context. Incorrect data type can cause undefined behaviour

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ContextRepr()

```
template<typename T >
ContextRepr< T >::ContextRepr (
            Context *const context )  [inline], [explicit]
```

Constructs new representation of a context.

Parameters

context	Pointer to the desired context to be represented

6.10.3 Member Function Documentation

6.10.3.1 operator=() [1/2]

```
template<typename T >
ContextRepr< T > & ContextRepr< T >::operator= (
            const T && data )  [inline]
```

Calls set(move) on the context with argument given as parameter. Updates data represented locally.

Parameters

data	Potential new data of context

Returns

Reference to this

6.10.3.2 operator=() [2/2]

```
template<typename T >
ContextRepr< T > & ContextRepr< T >::operator= (
            const T & data )  [inline]
```

Calls set(copy) on the context with argument given as parameter. Updates data represented locally.

Parameters

data	Potential new data of context

Returns

Reference to this

6.10.3.3 string()

```
template<typename T >
std::string ContextRepr< T >::string (
            const T & val ) const  [inline]
```

Equivalent to Context::string()

Parameters

val	Value to be converted

Returns

Result of string conversion

6.10.3.4 validate()

```
template<typename T >
bool ContextRepr< T >::validate (
            const T & potentialData ) const  [inline]
```

Equivalent to Context::validate()

Parameters

potentialData	Value for validation

Returns

True if the validation was successful

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

- NHF/Source/Core/AppData/ContextRepr.hpp

6.11 Controller Class Reference

Class for managing menus.

`#include <Controller.hpp>`

Public Member Functions

- Controller (Window &window)
  Constructs new empty Controller.
- void **renderPreview** ()
  Renders a preview. Supposed to be called before load, but not necessary.
- void **load** ()
  Builds menu tree.
- bool isActive ()
  Switches the current menu to another based on the its state.
- MenuBase * operator-> () const
  Returns pointer to the current menu.

6.11.1 Detailed Description

Class for managing menus.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Controller()

```
Controller::Controller (
            Window & window )  [explicit]
```

Constructs new empty Controller.

Parameters

window	The window of the application

6.11.3 Member Function Documentation

6.11.3.1 isActive()

```
bool Controller::isActive ( )
```

Switches the current menu to another based on the its state.

Returns

True if the menu current menu didn't request the application to be closed

6.11.3.2 operator->()

```
MenuBase * Controller::operator-> ( ) const
```

Returns pointer to the current menu.

Returns

Pointer to the current menu

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

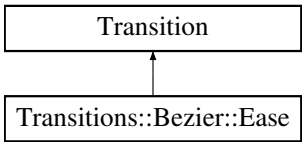
- NHF/Source/Core/Controller.hpp
- NHF/Source/Core/Controller.cpp

6.12 Transitions::Bezier::Ease Class Reference

Transition class for cubic Bezier easing.

```
#include <Transitions.hpp>
```

Inheritance diagram for Transitions::Bezier::Ease:



Public Member Functions

- Ease (Transitionable *object)

  Construct a new Ease.

Additional Inherited Members

6.12.1 Detailed Description

Transition class for cubic Bezier easing.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Ease()

```
Transitions::Bezier::Ease::Ease (
            Transitionable * object )  [explicit]
```

Construct a new Ease.

Parameters

object	The object of the transition

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

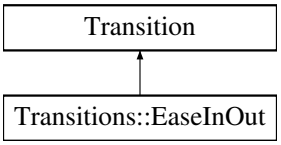
- NHF/Source/Utilities/Math/Transitions.hpp
- NHF/Source/Utilities/Math/Transitions.cpp

6.13 Transitions::EaseInOut Class Reference

Class for performing physics-like quadratic transitions (accelerate, then break)

```
#include <Transitions.hpp>
```

Inheritance diagram for Transitions::EaseInOut:



Public Member Functions

- EaseInOut (Transitionable *object)

  Constructs a new EaseInOut transition for the given object.
- bool start (const sf::Vector2f &distance, int time) override

  Starts the transition if it is not active.

Additional Inherited Members

6.13.1 Detailed Description

Class for performing physics-like quadratic transitions (accelerate, then break)

6.13.2 Constructor & Destructor Documentation

6.13.2.1 EaseInOut()

```
Transitions::EaseInOut::EaseInOut (
            Transitionable * object )  [explicit]
```

Constructs a new EaseInOut transition for the given object.

Parameters

object	

6.13.3 Member Function Documentation

6.13.3.1 start()

```
bool Transitions::EaseInOut::start (
            const sf::Vector2f & distance,
            int time )  [inline], [override], [virtual]
```

Starts the transition if it is not active.

Parameters

distance	The overall distance of the transition
time	The overall time of the transition

Returns

True if the transition could be started

Reimplemented from Transition.

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

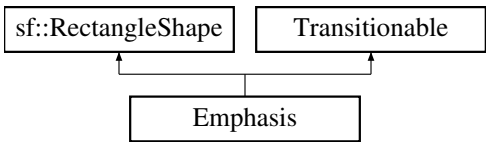
- NHF/Source/Utilities/Math/Transitions.hpp
- NHF/Source/Utilities/Math/Transitions.cpp

6.14 Emphasis Class Reference

Class for making sf::RectangleShape transitionable.

`#include <Bar.hpp>`

Inheritance diagram for Emphasis:



Additional Inherited Members

6.14.1 Detailed Description

Class for making sf::RectangleShape transitionable.

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

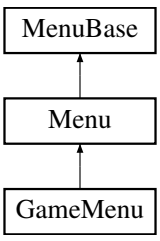
- NHF/Source/GUI/Widgets/Bar.hpp

6.15 GameMenu Class Reference

Class for managing and displaying the game menu of the application.

`#include <GameMenu.hpp>`

Inheritance diagram for GameMenu:



Public Member Functions

- **GameMenu** ()

  Constructs a new GameMenu.
- void handleEvent (const sf::Event &event) override

  Handles user input inside the game menu.
- void update () override

  Updates the states of the menu.
- void render () override

  Renders the contents of the menu to the screen.
- void init () override

  Initializes the states of the menu.
- void pause () override

  Pauses the menu's states.
- void resume () override

  Resumes the menu's states.

Additional Inherited Members

6.15.1 Detailed Description

Class for managing and displaying the game menu of the application.

6.15.2 Member Function Documentation

6.15.2.1 handleEvent()

```
void GameMenu::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input inside the game menu.

Implements MenuBase.

6.15.2.2 init()

```
void GameMenu::init ( )  [override], [virtual]
```

Initializes the states of the menu.

Implements MenuBase.

6.15.2.3 pause()

```
void GameMenu::pause ( )  [override], [virtual]
```

Pauses the menu's states.

Implements MenuBase.

6.15.2.4 render()

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

Renders the contents of the menu to the screen.

Implements MenuBase.

6.15.2.5 resume()

```
void GameMenu::resume ( )  [override], [virtual]
```

Resumes the menu's states.

Implements MenuBase.

6.15.2.6 update()

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

Updates the states of the menu.

Implements MenuBase.

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

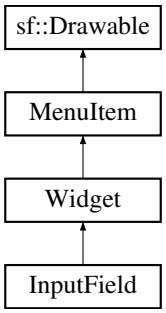
- NHF/Source/Menus/GameMenu.hpp
- NHF/Source/Menus/GameMenu.cpp

6.16 InputField Class Reference

Class that acts as an input field widget. Utilizes the context system under the hood.

`#include <InputField.hpp>`

Inheritance diagram for InputField:



Public Member Functions

- InputField (const sf::Font &fontStyle, unsigned characterSize, const std::string_view &contextName)
  Constructs a new InputField.
- void setPosition (const sf::Vector2f &position) override
  Setter for the input field's position.
- void handleEvent (const sf::Event &event) override
  Handles user input.
- void update () override
  Updates the input field.
- void init () override
  Initializes the input field.

Additional Inherited Members

6.16.1 Detailed Description

Class that acts as an input field widget. Utilizes the context system under the hood.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 InputField()

```
InputField::InputField (
            const sf::Font & fontStyle,
            unsigned characterSize,
            const std::string_view & contextName )
```

Constructs a new InputField.

Parameters

fontStyle	The style of the text displayed inside the input field
characterSize	The character size of the text
contextName	The name of the context that the class is hooked to

6.16.3 Member Function Documentation

6.16.3.1 handleEvent()

```
void InputField::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input.

Parameters

event	User input

Reimplemented from MenuItem.

6.16.3.2 init()

```
void InputField::init ( )  [override], [virtual]
```

Initializes the input field.

Reimplemented from MenuItem.

6.16.3.3 setPosition()

```
void InputField::setPosition (
            const sf::Vector2f & position )  [override], [virtual]
```

Setter for the input field's position.

Parameters

position	New position

Reimplemented from Widget.

6.16.3.4 update()

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

Updates the input field.

Reimplemented from MenuItem.

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

- NHF/Source/GUI/Widgets/InputField.hpp
- NHF/Source/GUI/Widgets/InputField.cpp

6.17 util::vector< T >::iterator Class Reference

iterator class for vector

`#include <vector.hpp>`

Public Member Functions

- iterator (T *vector, std::size_t index, std::size_t size)
  Constructs a new iterator for a specific vector.
- iterator & operator++ ()
  Pre-increments index stored inside the iterator.
- iterator operator++ (int)
  Post-increments index stored inside the iterator.
- bool operator== (const iterator &other) const
  Checks if the index of other is equal to the index of this.
- T & operator* ()
  Looks for the element stored inside vector at the indexes position.

6.17.1 Detailed Description

template<typename T>
class util::vector< T >::iterator

iterator class for vector

6.17.2 Constructor & Destructor Documentation

6.17.2.1 iterator()

```
template<typename T >
util::vector< T >::iterator::iterator (
            T * vector,
            std::size_t index,
            std::size_t size )  [inline], [explicit]
```

Constructs a new iterator for a specific vector.

Parameters

vector	Pointer to the first element of the vector
index	Index of the n^th element
size	Number of elements inside the vector

6.17.3 Member Function Documentation

6.17.3.1 operator*()

```
template<typename T >
T & util::vector< T >::iterator::operator* ( )  [inline]
```

Looks for the element stored inside vector at the indexes position.

Returns

Reference to element

6.17.3.2 operator++() [1/2]

```
template<typename T >
iterator & util::vector< T >::iterator::operator++ ( )  [inline]
```

Pre-increments index stored inside the iterator.

Returns

Reference to self

6.17.3.3 operator++() [2/2]

```
template<typename T >
iterator util::vector< T >::iterator::operator++ (
            int  )  [inline]
```

Post-increments index stored inside the iterator.

Parameters

* *	

Returns

Iterator equal to this before increment

6.17.3.4 operator==()

```
template<typename T >
bool util::vector< T >::iterator::operator== (
            const iterator & other ) const  [inline]
```

Checks if the index of other is equal to the index of this.

Parameters

other	Other iterator

Returns

True if the indexes are equal

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

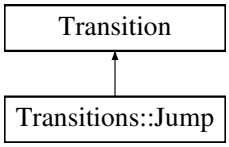
- NHF/Source/Utilities/STL/vector.hpp

6.18 Transitions::Jump Class Reference

Transition class for jumping transition.

`#include <Transitions.hpp>`

Inheritance diagram for Transitions::Jump:



Public Member Functions

- Jump (Transitionable *object)

  Constructs a new Jump.
- bool start (const sf::Vector2f &distance, int time) override

  Starts the transition if it is not active.

Additional Inherited Members

6.18.1 Detailed Description

Transition class for jumping transition.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Jump()

```
Transitions::Jump::Jump (
            Transitionable * object )  [explicit]
```

Constructs a new Jump.

Parameters

object	The object of the transition

6.18.3 Member Function Documentation

6.18.3.1 start()

```
bool Transitions::Jump::start (
            const sf::Vector2f & distance,
            int time )  [inline], [override], [virtual]
```

Starts the transition if it is not active.

Parameters

distance	The overall distance of the transition
time	The overall time of the transition

Returns

True if the transition could be started

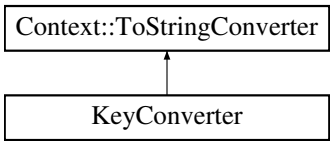
Reimplemented from Transition.

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

- NHF/Source/Utilities/Math/Transitions.hpp
- NHF/Source/Utilities/Math/Transitions.cpp

6.19 KeyConverter Class Reference

Inheritance diagram for KeyConverter:



Additional Inherited Members

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

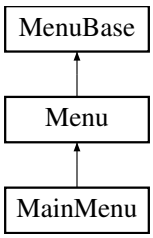
- NHF/Source/Core/AppData/ContextManager.cpp

6.20 MainMenu Class Reference

Class for managing and displaying the main menu of the application.

`#include <MainMenu.hpp>`

Inheritance diagram for MainMenu:



Public Member Functions

- **MainMenu** ()

  Constructs a new MainMenu.
- void handleEvent (const sf::Event &event) override

  Handles user input inside the main menu.
- void update () override

  Updates the states of the main menu.

Additional Inherited Members

6.20.1 Detailed Description

Class for managing and displaying the main menu of the application.

6.20.2 Member Function Documentation

6.20.2.1 handleEvent()

```
void MainMenu::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input inside the main menu.

Parameters

event	The user input

Implements MenuBase.

6.20.2.2 update()

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

Updates the states of the main menu.

Implements MenuBase.

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

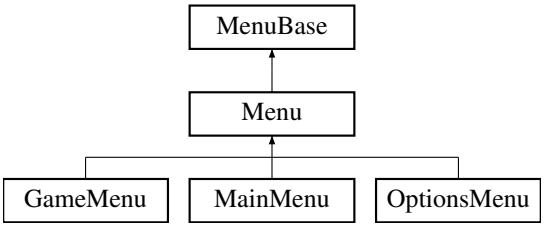
- NHF/Source/Menus/MainMenu.hpp
- NHF/Source/Menus/MainMenu.cpp

6.21 Menu Class Reference

Base class for the menus inside the application. Capable of holding MenuItems, making it easy to manage them.

```
#include <Menu.hpp>
```

Inheritance diagram for Menu:



Public Member Functions

- void handleEvent (const sf::Event &event) override

  Handles the user input for the managed items.
- void update () override

  Updates the managed items.
- void render () override

  Renders the managed items to the screen.
- void init () override

  Initializes the state of the menu and the managed items.
- void pause () override

  Pauses the menu and the managed items.
- void resume () override

  Resumes the menu and the managed items.

Protected Member Functions

- void addMenuItem (std::unique_ptr< MenuItem > item)

  Adds a MenuItem to be managed.
- bool isPaused () const

  Gets the paused flag of the menu.

6.21.1 Detailed Description

Base class for the menus inside the application. Capable of holding MenuItems, making it easy to manage them.

6.21.2 Member Function Documentation

6.21.2.1 addMenuItem()

```
void Menu::addMenuItem (
            std::unique_ptr< MenuItem > item )  [protected]
```

Adds a MenuItem to be managed.

Parameters

item	The MenuItem

6.21.2.2 handleEvent()

```
void Menu::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles the user input for the managed items.

Parameters

event	The user input

Implements MenuBase.

Reimplemented in OptionsMenu.

6.21.2.3 init()

```
void Menu::init ( )  [override], [virtual]
```

Initializes the state of the menu and the managed items.

Implements MenuBase.

6.21.2.4 isPaused()

```
bool Menu::isPaused ( ) const  [inline], [protected]
```

Gets the paused flag of the menu.

Returns

True of the menu is currently in paused mode

6.21.2.5 pause()

```
void Menu::pause ( )  [override], [virtual]
```

Pauses the menu and the managed items.

Implements MenuBase.

6.21.2.6 render()

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

Renders the managed items to the screen.

Implements MenuBase.

6.21.2.7 resume()

```
void Menu::resume ( )  [override], [virtual]
```

Resumes the menu and the managed items.

Implements MenuBase.

6.21.2.8 update()

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

Updates the managed items.

Implements MenuBase.

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

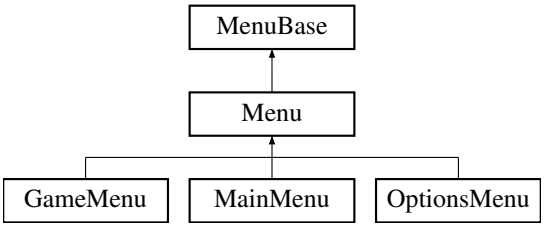
- NHF/Source/Menus/Menu.hpp
- NHF/Source/Menus/Menu.cpp

6.22 MenuBase Class Reference

Abstract Menu class. Manages state of Menu that can be accessed by the Controller class. This state is used to inform the controller when it needs to switch to a different menu or close the application.

```
#include <MenuBase.hpp>
```

Inheritance diagram for MenuBase:



Public Member Functions

- virtual void handleEvent (const sf::Event &event)=0

  Pure virtual for handling user input.
- virtual void update ()=0

  Pure virtual for refreshing the backend of the menu.
- virtual void render ()=0

  Pure virtual for refreshing the view.
- virtual void init ()=0

  Pure virtual for initializing the backend of the menu.
- virtual void pause ()=0

  Pure virtual for pausing the menu.
- virtual void resume ()=0

  Pure virtual for resuming the menu.
- virtual ~**MenuBase** ()=default

  Default virtual desctructor.

Protected Member Functions

- void open (const std::string_view &next)

  Sets state to represent the next menu to be opened.
- void **openLast** ()

  Sets "__openLast__" flag to true. Class will inform the controller that the last visited menu needs to be opened.
- void **close** ()

  Sets "__isClosing__" flag to true. Class will inform the controller that the current menu needs to be closed.
- void **exit** ()

  Sets "__isExiting__" flag to true. Class will inform the controller that the application has to exit.

6.22.1 Detailed Description

Abstract Menu class. Manages state of Menu that can be accessed by the Controller class. This state is used to inform the controller when it needs to switch to a different menu or close the application.

6.22.2 Member Function Documentation

6.22.2.1 handleEvent()

```
virtual void MenuBase::handleEvent (
            const sf::Event & event )  [pure virtual]
```

Pure virtual for handling user input.

Parameters

event	User input

Implemented in GameMenu, MainMenu, Menu, and OptionsMenu.

6.22.2.2 init()

```
virtual void MenuBase::init ( )  [pure virtual]
```

Pure virtual for initializing the backend of the menu.

Implemented in GameMenu, and Menu.

6.22.2.3 open()

```
void MenuBase::open (
            const std::string_view & next )  [inline], [protected]
```

Sets state to represent the next menu to be opened.

Parameters

next	Name of the next menu to be opened

6.22.2.4 pause()

```
virtual void MenuBase::pause ( )  [pure virtual]
```

Pure virtual for pausing the menu.

Implemented in GameMenu, and Menu.

6.22.2.5 render()

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

Pure virtual for refreshing the view.

Implemented in GameMenu, and Menu.

6.22.2.6 resume()

```
virtual void MenuBase::resume ( )  [pure virtual]
```

Pure virtual for resuming the menu.

Implemented in GameMenu, and Menu.

6.22.2.7 update()

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

Pure virtual for refreshing the backend of the menu.

Implemented in GameMenu, MainMenu, and Menu.

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

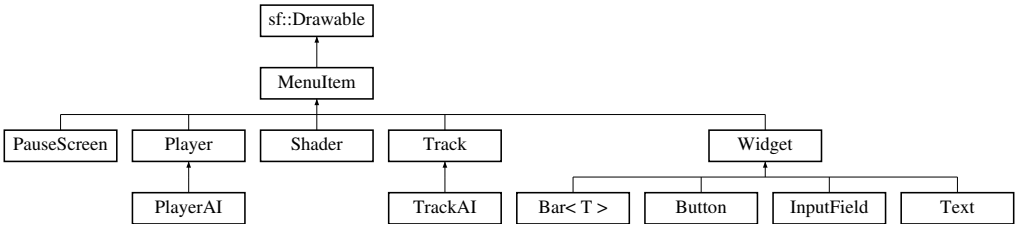
- NHF/Source/Core/Controller/MenuBase.hpp

6.23 MenuItem Class Reference

Abstract class for managing content inside the menus. Capable of managing the contents transitions.

`#include <MenuItem.hpp>`

Inheritance diagram for MenuItem:



Public Member Functions

- **MenuItem** ()=default

  Constructs a new MenuItem.
- MenuItem (const std::vector< Transition * > &transitions)

  Constructs a new MenuItem and adds transition to it for management.
- virtual void handleEvent (const sf::Event &event)

  Virtual function for making the class compatible with the Menu class.
- virtual void update ()

  Updates the managed transitions.
- virtual void init ()

  Initializes the managed transitions and sets the paused flag to false.
- virtual void pause ()

  Pauses the managed transitions and sets the paused flag to true.
- virtual void resume ()

  Resumes the managed transitions and sets the paused flag to false.

Protected Member Functions

- bool isPaused () const

  Checks whether the managed content is in paused mode.
- void addTransition (Transition *transition)

  Adds a transition to the MenuItem for management.

6.23.1 Detailed Description

Abstract class for managing content inside the menus. Capable of managing the contents transitions.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 MenuItem()

```
MenuItem::MenuItem (
            const std::vector< Transition * > & transitions )  [inline], [explicit]
```

Constructs a new MenuItem and adds transition to it for management.

Parameters

transitions	The transitions for management

6.23.3 Member Function Documentation

6.23.3.1 addTransition()

```
void MenuItem::addTransition (
            Transition * transition )  [inline], [protected]
```

Adds a transition to the MenuItem for management.

Parameters

transition	The transition

6.23.3.2 handleEvent()

```
virtual void MenuItem::handleEvent (
            const sf::Event & event )  [inline], [virtual]
```

Virtual function for making the class compatible with the Menu class.

Reimplemented in TrackAI, PlayerAI, Bar< T >, Button, InputField, PauseScreen, Player, and Track.

6.23.3.3 init()

```
void MenuItem::init ( )  [virtual]
```

Initializes the managed transitions and sets the paused flag to false.

Reimplemented in Bar< T >, InputField, PauseScreen, Player, and Track.

6.23.3.4 isPaused()

```
bool MenuItem::isPaused ( ) const  [inline], [protected]
```

Checks whether the managed content is in paused mode.

Returns

True if it is paused

6.23.3.5 pause()

```
void MenuItem::pause ( )  [virtual]
```

Pauses the managed transitions and sets the paused flag to true.

Reimplemented in Player.

6.23.3.6 resume()

```
void MenuItem::resume ( )  [virtual]
```

Resumes the managed transitions and sets the paused flag to false.

Reimplemented in Button, and Track.

6.23.3.7 update()

```
void MenuItem::update ( )  [virtual]
```

Updates the managed transitions.

Reimplemented in Bar< T >, InputField, Player, and Track.

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

- NHF/Source/Menus/MenuItem.hpp
- NHF/Source/Menus/MenuItem.cpp

6.24 MenuNode Class Reference

Class for storing a menu and information about its location inside the menu tree.

```
#include <MenuNode.hpp>
```

Public Member Functions

- **MenuNode** ()=default

  Constructs a new MenuNode storing nothing.
- MenuNode (std::unique_ptr< MenuBase > item, MenuNode *parent=nullptr)

  Constructs a new MenuNode storing a menu given as parameter and setting the parent node.
- void addChild (const std::string &name, std::unique_ptr< MenuBase > child)

  Tries appending child to locals.
- MenuBase * get () const

  Getter for the managed menu.
- MenuNode * getParent () const

  Getter for parent node.
- MenuNode * findChild (const std::string_view &name)

  Searches for child inside locals.
- MenuNode * getLastVisitedChild () const

  Getter for last visited child.
- void setLastVisitedChild (MenuNode *child)

  Setter for last visited child.

6.24.1 Detailed Description

Class for storing a menu and information about its location inside the menu tree.

@detailed Stores pointer to its parent, and stores children locally. Keeps track of last visited child

6.24.2 Constructor & Destructor Documentation

6.24.2.1 MenuNode()

```
MenuNode::MenuNode (
            std::unique_ptr< MenuBase > item,
            MenuNode * parent = nullptr )  [inline]
```

Constructs a new MenuNode storing a menu given as parameter and setting the parent node.

Parameters

item	Menu to be managed by node
parent	Parent node of class

6.24.3 Member Function Documentation

6.24.3.1 addChild()

```
void MenuNode::addChild (
            const std::string & name,
            std::unique_ptr< MenuBase > child )
```

Tries appending child to locals.

Parameters

name	Name of the child
child	Menu of the child

6.24.3.2 findChild()

```
MenuNode * MenuNode::findChild (
            const std::string_view & name )
```

Searches for child inside locals.

Parameters

name	Name of the desired child

Returns

Pointer to the child. Nullptr if the child is not found

6.24.3.3 get()

```
MenuBase * MenuNode::get ( ) const  [inline]
```

Getter for the managed menu.

Returns

Pointer to the managed menu

6.24.3.4 getLastVisitedChild()

```
MenuNode * MenuNode::getLastVisitedChild ( ) const  [inline]
```

Getter for last visited child.

Returns

Pointer to last visited child

6.24.3.5 getParent()

```
MenuNode * MenuNode::getParent ( ) const  [inline]
```

Getter for parent node.

Returns

Pointer to parent node

6.24.3.6 setLastVisitedChild()

```
void MenuNode::setLastVisitedChild (
            MenuNode * child )  [inline]
```

Setter for last visited child.

Parameters

child	Pointer to last visited child

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

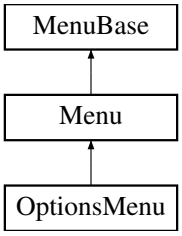
- NHF/Source/Core/Controller/MenuNode.hpp
- NHF/Source/Core/Controller/MenuNode.cpp

6.25 OptionsMenu Class Reference

Class for managing and displaying the options menu of the application.

`#include <OptionsMenu.hpp>`

Inheritance diagram for OptionsMenu:



Public Member Functions

- **OptionsMenu** ()

  Constructs new OptionsMenu.
- void handleEvent (const sf::Event &event) override

  Handles the user input inside the options menu.

Additional Inherited Members

6.25.1 Detailed Description

Class for managing and displaying the options menu of the application.

6.25.2 Member Function Documentation

6.25.2.1 handleEvent()

```
void OptionsMenu::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles the user input inside the options menu.

Parameters

event	The user input

Reimplemented from Menu.

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

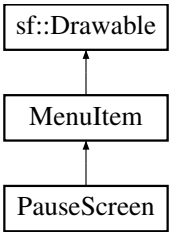
- NHF/Source/Menus/OptionsMenu.hpp
- NHF/Source/Menus/OptionsMenu.cpp

6.26 PauseScreen Class Reference

Class for displaying and managing the pause screen filter inside the game menu.

```
#include <PauseScreen.hpp>
```

Inheritance diagram for PauseScreen:



Public Member Functions

- PauseScreen (const std::function< void()> &resumeMenu, const std::function< void()> &closeMenu, const std::function< void()> &initMenu)

  Constructs new PauseScreen.
- void **gameOver** ()

  Changes the PauseScreen to reflect an end of game filter.
- void handleEvent (const sf::Event &event) override

  Handles user input.
- void init () override

  Initiates PauseScreen.

Additional Inherited Members

6.26.1 Detailed Description

Class for displaying and managing the pause screen filter inside the game menu.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 PauseScreen()

```
PauseScreen::PauseScreen (
            const std::function< void()> & resumeMenu,
            const std::function< void()> & closeMenu,
            const std::function< void()> & initMenu )  [explicit]
```

Constructs new PauseScreen.

Parameters

resumeMenu	Function that resumes the (game) menu
closeMenu	Function that closes the (game) menu
initMenu	Function that initiates the (game) menu

6.26.3 Member Function Documentation

6.26.3.1 handleEvent()

```
void PauseScreen::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input.

Parameters

event	User input

Reimplemented from MenuItem.

6.26.3.2 init()

```
void PauseScreen::init ( )  [override], [virtual]
```

Initiates PauseScreen.

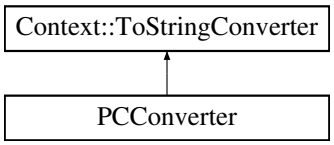
Reimplemented from MenuItem.

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

- NHF/Source/Menus/GameMenu/PauseScreen.hpp
- NHF/Source/Menus/GameMenu/PauseScreen.cpp

6.27 PCConverter Class Reference

Inheritance diagram for PCConverter:



Additional Inherited Members

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

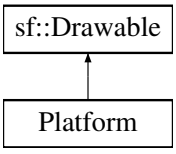
- NHF/Source/Core/AppData/ContextManager.cpp

6.28 Platform Class Reference

Class that displays and manages a platform.

```
#include <Platform.hpp>
```

Inheritance diagram for Platform:



Public Member Functions

- Platform (const PreCalculator &preCalc, float rotation, float width, unsigned speed)

  Constructs a new Platform.
- float getInnerRadius () const

  Getter for inner radius.
- float getOuterRadius () const

  Getter for outer radius.
- float getRotation () const

  Getter for rotation.
- float getWidth () const

  Getter for width.
- bool isInside (const PolarVector &point) const

  Check whether a point is on the platform.
- void rotate (float angle)

  Rotates the platform by the amount given as param.
- void **update** ()

  Updates the platform.
- bool isDead () const

  Checks whether the platform is still inside the area of the window.

Static Public Member Functions

- static void setOrigin (const sf::Vector2f &origin)

  Setter for _origin.

6.28.1 Detailed Description

Class that displays and manages a platform.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 Platform()

```
Platform::Platform (
            const PreCalculator & preCalc,
            float rotation,
            float width,
            unsigned speed )
```

Constructs a new Platform.

Parameters

preCalc	Reference to a pre-calculator
rotation	Initial rotation of the platform (in radians)
width	Width of the platform (in radians)
speed	The amount of times a platform needs to be updated to move by a range of 1 platform

6.28.3 Member Function Documentation

6.28.3.1 getInnerRadius()

```
float Platform::getInnerRadius ( ) const  [inline]
```

Getter for inner radius.

Returns

The inner radius of the platform

6.28.3.2 getOuterRadius()

```
float Platform::getOuterRadius ( ) const  [inline]
```

Getter for outer radius.

Returns

The outer radius of the platform

6.28.3.3 getRotation()

```
float Platform::getRotation ( ) const  [inline]
```

Getter for rotation.

Returns

The rotation of the platform in radians

6.28.3.4 getWidth()

```
float Platform::getWidth ( ) const  [inline]
```

Getter for width.

Returns

The width of the platform in radians

6.28.3.5 isDead()

```
bool Platform::isDead ( ) const
```

Checks whether the platform is still inside the area of the window.

Returns

True if the platform is outside

6.28.3.6 isInside()

```
bool Platform::isInside (
            const PolarVector & point ) const
```

Check whether a point is on the platform.

Parameters

point	The point in question

Returns

True if the point is on the platform

6.28.3.7 rotate()

```
void Platform::rotate (
            float angle )
```

Rotates the platform by the amount given as param.

Parameters

angle	The amount that will be added to the platforms rotation (in radians)

6.28.3.8 setOrigin()

```
void Platform::setOrigin (
            const sf::Vector2f & origin )  [static]
```

Setter for _origin.

Parameters

origin	The new origin

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

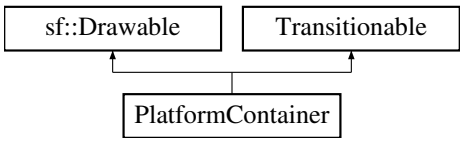
- NHF/Source/Menus/GameMenu/Platform.hpp
- NHF/Source/Menus/GameMenu/Platform.cpp

6.29 PlatformContainer Class Reference

Class for managing platforms.

`#include <PlatformContainer.hpp>`

Inheritance diagram for PlatformContainer:



Public Member Functions

- PlatformContainer (const PreCalculator &preCalc)

  Constructs a new platform container.
- float getPlatformWidth () const

  Gets the width of the individual platforms.
- bool isInside (const PolarVector &point) const

  Checks whether a point is on the platforms.
- bool AI_help (PolarVector playerFeet, int &switchingState)

  A helper function for the algorithm inside the main menu. Checks when the algorithmic player needs to jump, and sets switchingState for the Track class.
- void rotate (float angle)

  Rotates all platforms by a given angle.
- void transition (const sf::Vector2f &amount) override

  Calls rotate with the X coordinate of the given vector.
- void **update** ()

  Updates the state of the platforms.
- void init (unsigned laneCount, unsigned speed)

  Initializes the platforms.

6.29.1 Detailed Description

Class for managing platforms.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 PlatformContainer()

```
PlatformContainer::PlatformContainer (
            const PreCalculator & preCalc )  [explicit]
```

Constructs a new platform container.

Parameters

preCalc	Reference to a PreCalculator

6.29.3 Member Function Documentation

6.29.3.1 AI_help()

```
bool PlatformContainer::AI_help (
            PolarVector playerFeet,
            int & switchingState )
```

A helper function for the algorithm inside the main menu. Checks when the algorithmic player needs to jump, and sets switchingState for the Track class.

Parameters

playerFeet	The coordinates of the player
switchingState	Reference to the Track's switching state

Returns

True if the algorithm has to jump

6.29.3.2 getPlatformWidth()

```
float PlatformContainer::getPlatformWidth ( ) const  [inline]
```

Gets the width of the individual platforms.

Returns

The platform's width

6.29.3.3 init()

```
void PlatformContainer::init (
            unsigned laneCount,
            unsigned speed )
```

Initializes the platforms.

Parameters

laneCount	The number of lanes for the platforms to form
speed	The amount of times a platform needs to be updated to move by a range of 1 platform

6.29.3.4 isInside()

```
bool PlatformContainer::isInside (
            const PolarVector & point ) const
```

Checks whether a point is on the platforms.

Parameters

point	The point in question

Returns

True if it is on the platforms

6.29.3.5 rotate()

```
void PlatformContainer::rotate (
            float angle )
```

Rotates all platforms by a given angle.

Parameters

angle	The given angle

6.29.3.6 transition()

```
void PlatformContainer::transition (
```

```
const sf::Vector2f & amount )  [inline], [override], [virtual]
```

Calls rotate with the X coordinate of the given vector.

Parameters

amount	The given vector

Implements Transitionable.

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

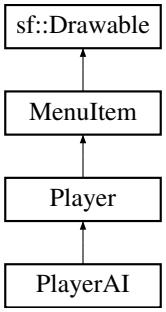
- NHF/Source/Menus/GameMenu/PlatformContainer.hpp
- NHF/Source/Menus/GameMenu/PlatformContainer.cpp

6.30 Player Class Reference

Class that acts as a player inside the game.

`#include <Player.hpp>`

Inheritance diagram for Player:



Public Member Functions

- **Player** ()

  Constructs a new player.
- bool isJumping () const

  Checks whether the player is in a jumping state.
- sf::Vector2f getFeet () const

  Gets the coordinates of the player's feet.
- void setPosition (const sf::Vector2f &feet)

  Sets the player's position.
- void handleEvent (const sf::Event &event) override

  Handles user input.
- void update () override

  Updates the state of the player.
- void init () override

  Initializes the Player class.
- void pause () override

  Sets the player's state to paused.

Protected Member Functions

- void **startJump** ()

  Starts the jumping process of the player.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Draws the player to the render target.

6.30.1 Detailed Description

Class that acts as a player inside the game.

6.30.2 Member Function Documentation

6.30.2.1 draw()

```
void Player::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [inline], [override], [protected]
```

Draws the player to the render target.

Parameters

target	The render target
states	The render states

6.30.2.2 getFeet()

```
sf::Vector2f Player::getFeet ( ) const  [inline]
```

Gets the coordinates of the player's feet.

Returns

The coordinate of the player's feet

6.30.2.3 handleEvent()

```
void Player::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input.

Parameters

event	The user input

Reimplemented from MenuItem.

Reimplemented in PlayerAI.

6.30.2.4 init()

```
void Player::init ( )  [override], [virtual]
```

Initializes the Player class.

Reimplemented from MenuItem.

6.30.2.5 isJumping()

```
bool Player::isJumping ( ) const  [inline]
```

Checks whether the player is in a jumping state.

Returns

True if the player is currently jumping

6.30.2.6 pause()

```
void Player::pause ( )  [override], [virtual]
```

Sets the player's state to paused.

Reimplemented from MenuItem.

6.30.2.7 setPosition()

```
void Player::setPosition (
            const sf::Vector2f & feet )
```

Sets the player's position.

Parameters

feet	The new coordinates of the player's feet

6.30.2.8 update()

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

Updates the state of the player.

Reimplemented from MenuItem.

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

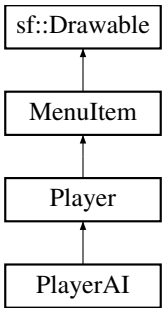
- NHF/Source/Menus/GameMenu/Player.hpp
- NHF/Source/Menus/GameMenu/Player.cpp

6.31 PlayerAI Class Reference

Class for managing the player in the main menu's background.

```
#include <MainMenu.hpp>
```

Inheritance diagram for PlayerAI:



Public Member Functions

- void **jump** ()

  Starts the jumping process of the player.
- void handleEvent (const sf::Event &) override

  Overrides the Player's handleEvent so that it does not reflect user input.

Additional Inherited Members

6.31.1 Detailed Description

Class for managing the player in the main menu's background.

6.31.2 Member Function Documentation

6.31.2.1 handleEvent()

```
void PlayerAI::handleEvent (
            const sf::Event &  )  [inline], [override], [virtual]
```

Overrides the Player's handleEvent so that it does not reflect user input.

Parameters

User	input

Reimplemented from Player.

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

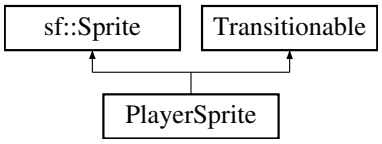
- NHF/Source/Menus/MainMenu.hpp

6.32 PlayerSprite Class Reference

Class for managing the sprite of the Player class.

```
#include <Player.hpp>
```

Inheritance diagram for PlayerSprite:



Public Member Functions

- **PlayerSprite** ()

  Constructs a new PlayerSprite, setting its initial position to {0,0}.
- void setInitPos (const sf::Vector2f &pos)

  Sets the initial position of the player.
- const sf::Vector2f & getInitPos () const

  Gets the initial position of the player.
- void transition (const sf::Vector2f &amount) override

  Moves the sprite by a given amount.
- void **init** ()

  Resets the sprite's position to its initial position.

6.32.1 Detailed Description

Class for managing the sprite of the Player class.

6.32.2 Member Function Documentation

6.32.2.1 getInitPos()

```
const sf::Vector2f & PlayerSprite::getInitPos ( ) const  [inline]
```

Gets the initial position of the player.

Returns

The initial position of the player

6.32.2.2 setInitPos()

```
void PlayerSprite::setInitPos (
            const sf::Vector2f & pos )  [inline]
```

Sets the initial position of the player.

Parameters

pos	The initial position

6.32.2.3 transition()

```
void PlayerSprite::transition (
            const sf::Vector2f & amount )  [inline], [override], [virtual]
```

Moves the sprite by a given amount.

Parameters

amount	The given amount

Implements Transitionable.

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

- NHF/Source/Menus/GameMenu/Player.hpp
- NHF/Source/Menus/GameMenu/Player.cpp

6.33 PolarVector Struct Reference

Struct for representing a vector in polar coordinates.

`#include <PolarVector.hpp>`

Public Member Functions

- **PolarVector** (float radius_=0, float angle_=0)

Public Attributes

- float **radius**
- float **angle**

6.33.1 Detailed Description

Struct for representing a vector in polar coordinates.

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

- NHF/Source/Utilities/Math/PolarVector.hpp

6.34 PreCalculator Class Reference

Class for performing pre-calculations.

`#include <PreCalculator.hpp>`

Public Member Functions

- **PreCalculator** ()

  Constructs a new PreCalculator and performs the calculations.
- const PolarVector & getPolarVector (const sf::Vector2f &vector) const

  Gets the pre-calculated result of converting a vector given in Descartes coordinates to a polar vector.

6.34.1 Detailed Description

Class for performing pre-calculations.

6.34.2 Member Function Documentation

6.34.2.1 getPolarVector()

```
const PolarVector & PreCalculator::getPolarVector (
            const sf::Vector2f & vector ) const  [inline]
```

Gets the pre-calculated result of converting a vector given in Descartes coordinates to a polar vector.

Parameters

vector	The vector to be converted in Descartes coordinates

Returns

The polar result of the conversion

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

- NHF/Source/Menus/GameMenu/PreCalculator.hpp
- NHF/Source/Menus/GameMenu/PreCalculator.cpp

6.35 PreView Class Reference

Class for rendering a loading screen.

`#include <PreView.hpp>`

Static Public Member Functions

- static void **render** ()

  Renders loading screen.

6.35.1 Detailed Description

Class for rendering a loading screen.

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

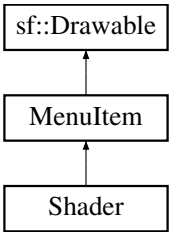
- NHF/Source/Core/Controller/PreView.hpp
- NHF/Source/Core/Controller/PreView.cpp

6.36 Shader Class Reference

Class for drawing a dark filter, making the contents in the background easy to distinguish opposed to the main contents of the menu.

`#include <MainMenu.hpp>`

Inheritance diagram for Shader:



Public Member Functions

- **Shader** ()

  Constructs a new Shader.

Additional Inherited Members

6.36.1 Detailed Description

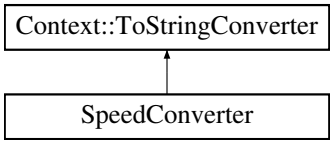
Class for drawing a dark filter, making the contents in the background easy to distinguish opposed to the main contents of the menu.

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

- NHF/Source/Menus/MainMenu.hpp

6.37 SpeedConverter Class Reference

Inheritance diagram for SpeedConverter:



Additional Inherited Members

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

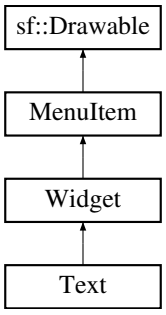
- NHF/Source/Core/AppData/ContextManager.cpp

6.38 Text Class Reference

Class that acts as a text widget.

`#include <Text.hpp>`

Inheritance diagram for Text:



Public Member Functions

- **Text** ()=default

  Constructs an empty text.
- Text (const sf::String &text, const sf::Font &fontStyle, unsigned characterSize)

  Constructs a new text.
- **Text** (const Text &)=default

  Copy constructor of the class.
- void setPosition (const sf::Vector2f &position) override

  Sets the position of the text.
- void setFillColor (const sf::Color &color)

  Sets the fill color of the text.
- void setString (const sf::String &string)

  Sets the displayed message.
- const sf::String & getString () const

  Gets the displayed message.

Additional Inherited Members

6.38.1 Detailed Description

Class that acts as a text widget.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 Text()

```
Text::Text (
            const sf::String & text,
            const sf::Font & fontStyle,
            unsigned characterSize )
```

Constructs a new text.

Parameters

text	Message that the class is going to display
fontStyle	Font style of the text
characterSize	Character size of the text

6.38.3 Member Function Documentation

6.38.3.1 getString()

```
const sf::String & Text::getString ( ) const  [inline]
```

Gets the displayed message.

Returns

The displayed message

6.38.3.2 setFillColor()

```
void Text::setFillColor (
            const sf::Color & color )  [inline]
```

Sets the fill color of the text.

Parameters

color	The new fill color

6.38.3.3 setPosition()

```
void Text::setPosition (
            const sf::Vector2f & position )  [inline], [override], [virtual]
```

Sets the position of the text.

Parameters

position	The new position

Reimplemented from Widget.

6.38.3.4 setString()

```
void Text::setString (
            const sf::String & string )
```

Sets the displayed message.

Parameters

string	The new message

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

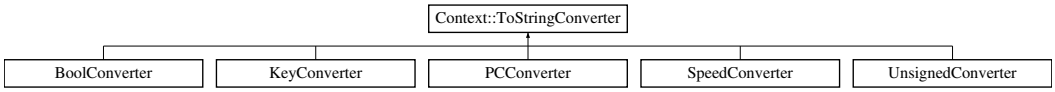
- NHF/Source/GUI/Widgets/Text.hpp
- NHF/Source/GUI/Widgets/Text.cpp

6.39 Context::ToStringConverter Class Reference

Functor for converting Context data to string.

`#include <Context.hpp>`

Inheritance diagram for Context::ToStringConverter:



Public Member Functions

- explicit (false) ToStringConverter(const std
  Constructs new converter.
- std::string operator() (const std::any &val) const
  Calls function stored inside class.

6.39.1 Detailed Description

Functor for converting Context data to string.

6.39.2 Member Function Documentation

6.39.2.1 explicit()

```
Context::ToStringConverter::explicit (
            false  ) const  [inline]
```

Constructs new converter.

Parameters

func	Inside-function of the functor class

6.39.2.2 operator()()

```
std::string Context::ToStringConverter::operator() (
            const std::any & val ) const  [inline]
```

Calls function stored inside class.

Parameters

val	Value to be given to function as param

Returns

Result of calling the inside-function

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

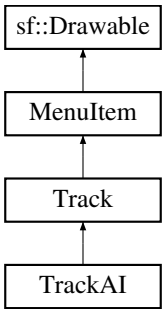
- NHF/Source/Core/AppData/Context.hpp

6.40 Track Class Reference

Class for managing the playing field of the game.

`#include <Track.hpp>`

Inheritance diagram for Track:



Public Member Functions

- **Track** ()

  Constructs a new Track.
- bool isOnPlatforms (const sf::Vector2f &point) const

  Checks whether a point is on one of the platforms inside the playing field.
- bool AI_jump (const sf::Vector2f &playerFeet)

  Helper function for the background algorithm inside the main menu.
- void handleEvent (const sf::Event &event) override

  Handles user input.
- void update () override

  Updates the state of the playing field.
- void init () override

  Initializes the states of the playing field.
- void resume () override

  Sets the state of the Track to resumed.

Protected Member Functions

- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Draws the playing field to the render target.
- void handleMouseEvent (const sf::Event &mouseEvent)

  Handles mouse events.
- void handleKeyEvent (const sf::Event &keyEvent)

  Handles key events.

6.40.1 Detailed Description

Class for managing the playing field of the game.

6.40.2 Member Function Documentation

6.40.2.1 AI_jump()

```
bool Track::AI_jump (
            const sf::Vector2f & playerFeet )
```

Helper function for the background algorithm inside the main menu.

Parameters

playerFeet	The coordinates of the player's feet

Returns

True if the player needs to jump

6.40.2.2 draw()

```
void Track::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override], [protected]
```

Draws the playing field to the render target.

Parameters

target	The render target
states	The render states

6.40.2.3 handleEvent()

```
void Track::handleEvent (
            const sf::Event & event )  [override], [virtual]
```

Handles user input.

Parameters

event	The user input

Reimplemented from MenuItem.

Reimplemented in TrackAI.

6.40.2.4 handleKeyEvent()

```
void Track::handleKeyEvent (
            const sf::Event & keyEvent )  [protected]
```

Handles key events.

Parameters

mouseEvent	The key event to be handled

6.40.2.5 handleMouseEvent()

```
void Track::handleMouseEvent (
            const sf::Event & mouseEvent )  [protected]
```

Handles mouse events.

Parameters

mouseEvent	The mouse event to be handled

6.40.2.6 init()

```
void Track::init ( )  [override], [virtual]
```

Initializes the states of the playing field.

Reimplemented from MenuItem.

6.40.2.7 isOnPlatforms()

```
bool Track::isOnPlatforms (
            const sf::Vector2f & point ) const
```

Checks whether a point is on one of the platforms inside the playing field.

Parameters

point	The point in question

Returns

True if it is on a platform

6.40.2.8 resume()

```
void Track::resume ( )  [override], [virtual]
```

Sets the state of the Track to resumed.

Reimplemented from MenuItem.

6.40.2.9 update()

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

Updates the state of the playing field.

Reimplemented from MenuItem.

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

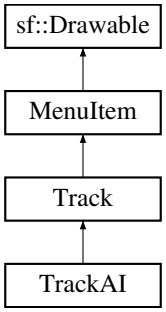
- NHF/Source/Menus/GameMenu/Track.hpp
- NHF/Source/Menus/GameMenu/Track.cpp

6.41 TrackAI Class Reference

Class for managing the track in the main menu's background.

```
#include <MainMenu.hpp>
```

Inheritance diagram for TrackAI:



Public Member Functions

- void handleEvent (const sf::Event &) override

  Overrides the Track's handleEvent() so that it does not reflect user input.

Additional Inherited Members

6.41.1 Detailed Description

Class for managing the track in the main menu's background.

6.41.2 Member Function Documentation

6.41.2.1 handleEvent()

```
void TrackAI::handleEvent (
            const sf::Event &  )  [inline], [override], [virtual]
```

Overrides the Track's handleEvent() so that it does not reflect user input.

Parameters

User	input

Reimplemented from Track.

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

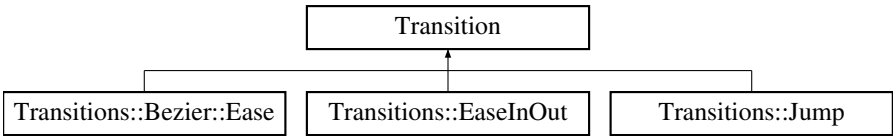
- NHF/Source/Menus/MainMenu.hpp

6.42 Transition Class Reference

Class for performing transition on a given.

`#include <Transitions.hpp>`

Inheritance diagram for Transition:



Public Member Functions

- bool isActive () const

  Checks whether the transition is still happening.
- virtual bool start (const sf::Vector2f &distance, int time)

  Starts the transition if it is not active.
- void **update** ()

  Updates the transition's process and does the transitioning.
- virtual void **init** ()

  Initializes the Transition, making it inactive.
- virtual void **pause** ()

  Paused the transition.
- virtual void **resume** ()

  Resumes the transition.
- virtual ~**Transition** ()=default

  Default virtual destructor.

Protected Member Functions

- Transition (Transitionable *object, const std::function< sf::Vector2f(int elapsedTime)> &getProgression, const std::function< sf::Vector2f()> &correctDistance=nullptr)

  Constructs a new Transiton.
- int getDurationTime () const

  Gets the duration time of the transition.
- const sf::Vector2f & getDurationDistance () const

  Gets the duration distance of the transition.
- int getElapsedTime () const

  Gets the time elapsed since the beginning of the current transition.
- const sf::Vector2f & getDistanceTraveled () const

  Gets the extent of the distance travelled since the beginning of the current transition.

6.42.1 Detailed Description

Class for performing transition on a given.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 Transition()

```
Transition::Transition (
            Transitionable * object,
            const std::function< sf::Vector2f(int elapsedTime)> & getProgression,
            const std::function< sf::Vector2f()> & correctDistance = nullptr )  [explicit],
[protected]
```

Constructs a new Transiton.

Parameters

object	The object of the transition
getProgression	Function to calculate the progress of the transition from 0 to t interval
correctDistance	Function to correct the left-over transition amount from 0 to t

6.42.3 Member Function Documentation

6.42.3.1 getDistanceTraveled()

```
const sf::Vector2f & Transition::getDistanceTraveled ( ) const  [inline], [protected]
```

Gets the extent of the distance travelled since the beginning of the current transition.

Returns

The travelled distance

6.42.3.2 getDurationDistance()

```
const sf::Vector2f & Transition::getDurationDistance ( ) const  [inline], [protected]
```

Gets the duration distance of the transition.

Returns

The duration distance of the transition

6.42.3.3 getDurationTime()

```
int Transition::getDurationTime ( ) const  [inline], [protected]
```

Gets the duration time of the transition.

Returns

The duration time of the transition

6.42.3.4 getElapsedTime()

```
int Transition::getElapsedTime ( ) const  [inline], [protected]
```

Gets the time elapsed since the beginning of the current transition.

Returns

The elapsed time

6.42.3.5 isActive()

```
bool Transition::isActive ( ) const  [inline]
```

Checks whether the transition is still happening.

Returns

True if it the object is in transition mode

6.42.3.6 start()

```
virtual bool Transition::start (
            const sf::Vector2f & distance,
            int time )  [inline], [virtual]
```

Starts the transition if it is not active.

Parameters

distance	The overall distance of the transition
time	The overall time of the transition

Returns

True if the transition could be started

Reimplemented in Transitions::EaseInOut, and Transitions::Jump.

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

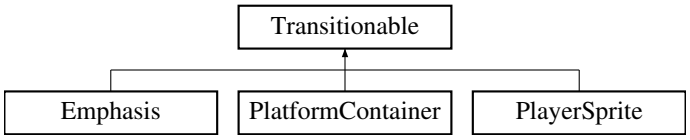
- NHF/Source/Utilities/Math/Transitions.hpp
- NHF/Source/Utilities/Math/Transitions.cpp

6.43 Transitionable Class Reference

Abstract class for making objects compatible with the Transition class.

`#include <Transitionable.hpp>`

Inheritance diagram for Transitionable:



Public Member Functions

- virtual void transition (const sf::Vector2f &amount)=0

6.43.1 Detailed Description

Abstract class for making objects compatible with the Transition class.

6.43.2 Member Function Documentation

6.43.2.1 transition()

```
virtual void Transitionable::transition (
            const sf::Vector2f & amount )  [pure virtual]
```

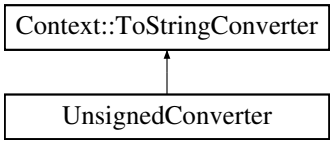
Implemented in PlatformContainer, and PlayerSprite.

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

- NHF/Source/Utilities/Math/Transitionable.hpp

6.44 UnsignedConverter Class Reference

Inheritance diagram for UnsignedConverter:



Additional Inherited Members

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

- NHF/Source/Core/AppData/ContextManager.cpp

6.45 Context::Validator Class Reference

Functor for validating Context data.

`#include <Context.hpp>`

Public Member Functions

- explicit (false) Validator(const std
  Constructs new validator.
- bool operator() (const std::any &data) const
  Calls function stored inside class.

6.45.1 Detailed Description

Functor for validating Context data.

6.45.2 Member Function Documentation

6.45.2.1 explicit()

```
Context::Validator::explicit (
            false  ) const  [inline]
```

Constructs new validator.

Parameters

func	Inside-function of the functor class

6.45.2.2 operator()()

```
bool Context::Validator::operator() (
            const std::any & data ) const  [inline]
```

Calls function stored inside class.

Parameters

val	Value to be given to function as param

Returns

True if data is valid

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

- NHF/Source/Core/AppData/Context.hpp

6.46 util::vector< T > Class Template Reference

Container class for imitating std::vector. This implementation is not error prone, use wisely! Add new features when necessary.

`#include <vector.hpp>`

Classes

- class iterator

  iterator class for vector

Public Member Functions

- **vector** ()=default

  Constructs a new vector that stores zero elements.
- **vector** (const vector &)=delete
- vector & **operator=** (const vector &)=delete
- template<typename... Args>
  void emplace_back (Args &&...args)

  Appends a new element to the end of the container. The element is constructed in place.
- iterator begin ()

  Returns an iterator to the first element of the vector. If the vector is empty, the returned iterator will be equal to end().
- iterator end ()

  Returns an iterator to the element following the last element of the vector.

6.46.1 Detailed Description

template<typename T>
class util::vector< T >

Container class for imitating std::vector. This implementation is not error prone, use wisely! Add new features when necessary.

Template Parameters

T	Type of the values stored inside the class

6.46.2 Member Function Documentation

6.46.2.1 begin()

```
template<typename T >
iterator util::vector< T >::begin ( )  [inline]
```

Returns an iterator to the first element of the vector. If the vector is empty, the returned iterator will be equal to end().

Returns

Iterator to the first element.

6.46.2.2 emplace_back()

```
template<typename T >
template<typename...  Args>
void util::vector< T >::emplace_back (
            Args &&...  args )  [inline]
```

Appends a new element to the end of the container. The element is constructed in place.

Template Parameters

...Args	Types of arguments given as parameters

Parameters

...args	Arguments to forward to the constructor of the element

6.46.2.3 end()

```
template<typename T >
iterator util::vector< T >::end ( )  [inline]
```

Returns an iterator to the element following the last element of the vector.

Returns

Iterator to the element following the last element.

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

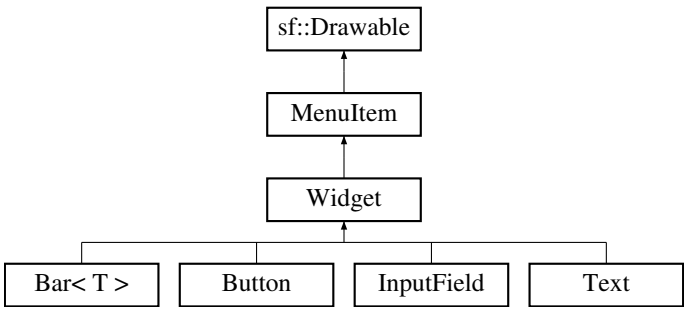
- NHF/Source/Utilities/STL/vector.hpp

6.47 Widget Class Reference

Abstract class for implementing widgets.

`#include <Widget.hpp>`

Inheritance diagram for Widget:



Public Member Functions

- Widget (const sf::Vector2f &position={ 0, 0 }, const sf::Vector2f &size={ 0, 0 })
  Constructs a new Widget.
- **Widget** (const Widget &other)=default
  Copy constructor.
- virtual void setPosition (const sf::Vector2f &position)
  Sets the position of the widget.
- const sf::Vector2f & getPosition () const
  Gets the position of the widget.
- const sf::Vector2f & getSize () const
  Gets the size of the widget.
- sf::FloatRect getLocalBounds () const
  Gets the bounds of the widget.
- virtual void center (const sf::FloatRect &frame)
  Centers the widget in the frame given as parameter.
- virtual void move (const sf::Vector2f &amount)
  Moves the widget by an given as parameter amount.

Protected Member Functions

- void setSize (const sf::Vector2f &size)

  Sets the size of the widget.

6.47.1 Detailed Description

Abstract class for implementing widgets.

6.47.2 Constructor & Destructor Documentation

6.47.2.1 Widget()

```
Widget::Widget (
            const sf::Vector2f & position = { 0, 0 },
            const sf::Vector2f & size = { 0, 0 } )  [inline]
```

Constructs a new Widget.

Parameters

position	The position of the new widget
size	The size of the new widget

6.47.3 Member Function Documentation

6.47.3.1 center()

```
virtual void Widget::center (
            const sf::FloatRect & frame )  [inline], [virtual]
```

Centers the widget in the frame given as parameter.

Parameters

frame	A frame where the widget will be centered

6.47.3.2 getLocalBounds()

```
sf::FloatRect Widget::getLocalBounds ( ) const  [inline]
```

Gets the bounds of the widget.

Returns

The bounds of the widget

6.47.3.3 getPosition()

```
const sf::Vector2f & Widget::getPosition ( ) const  [inline]
```

Gets the position of the widget.

Returns

The position of the widget

6.47.3.4 getSize()

```
const sf::Vector2f & Widget::getSize ( ) const  [inline]
```

Gets the size of the widget.

Returns

The size of the widget

6.47.3.5 move()

```
virtual void Widget::move (
            const sf::Vector2f & amount )  [inline], [virtual]
```

Moves the widget by an given as parameter amount.

Parameters

amount	The amount that the widget will move

6.47.3.6 setPosition()

```
virtual void Widget::setPosition (
            const sf::Vector2f & position )  [inline], [virtual]
```

Sets the position of the widget.

Parameters

position	The new position

Reimplemented in Bar< T >, Button, InputField, and Text.

6.47.3.7 setSize()

```
void Widget::setSize (
            const sf::Vector2f & size )  [inline], [protected]
```

Sets the size of the widget.

Parameters

size	The new size

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

- NHF/Source/GUI/Widget.hpp

6.48 Window Class Reference

Class for displaying the view. Adapter for sf::RenderWindow.

```
#include <Window.hpp>
```

Public Member Functions

- **Window** ()

  Constructs a new Window.
- void **open** () const

  Opens the window.
- void **close** () const

  Closes the window.

- bool isOpen () const

  Checks whether the window is open.
- bool pollEvent (sf::Event &event) const

  Calls pollEvent on the inner sf::RenderWindow.
- void lockFPS (int FPS=60)

  Locks the number of frames per second. Makes sure by causing potential delay that the right amount of time has passed since the last call of the function.

Static Public Member Functions

- static const sf::RenderWindow & window ()

  Getter for inner window.
- static sf::Vector2f getSize ()

  Getter for the size of the window.
- static sf::FloatRect getLocalBounds ()

  Getter for the bounds of the window.
- static void **clear** ()

  Clears the window.
- static void draw (const sf::Drawable &drawable, const sf::RenderStates &states=sf::RenderStates::Default)

  Draws object to window.
- static void **display** ()

  Displays the contents of the window.

6.48.1 Detailed Description

Class for displaying the view. Adapter for sf::RenderWindow.

6.48.2 Member Function Documentation

6.48.2.1 draw()

```
void Window::draw (
            const sf::Drawable & drawable,
            const sf::RenderStates & states = sf::RenderStates::Default )  [static]
```

Draws object to window.

Parameters

drawable	Object to be drawn
states	Render states

6.48.2.2 getLocalBounds()

```
sf::FloatRect Window::getLocalBounds ( )  [static]
```

Getter for the bounds of the window.

Returns

The bounds of the window

6.48.2.3 getSize()

```
sf::Vector2f Window::getSize ( )  [static]
```

Getter for the size of the window.

Returns

The size of the window

6.48.2.4 isOpen()

```
bool Window::isOpen ( ) const
```

Checks whether the window is open.

Returns

True if the window is open

6.48.2.5 lockFPS()

```
void Window::lockFPS (
             int FPS = 60 )
```

Locks the number of frames per second. Makes sure by causing potential delay that the right amount of time has passed since the last call of the function.

Parameters

FPS	Frames per second. Default is 60

6.48.2.6 pollEvent()

```
bool Window::pollEvent (
            sf::Event & event ) const
```

Calls pollEvent on the inner sf::RenderWindow.

Parameters

event	Reference to event where information will be distributed

Returns

True in case a new event was polled

6.48.2.7 window()

```
static const sf::RenderWindow & Window::window ( )  [inline], [static]
```

Getter for inner window.

Returns

Inner window

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

- NHF/Source/Core/Window.hpp
- NHF/Source/Core/Window.cpp

Chapter 7

File Documentation

7.1 App.hpp

```
1 #pragma once
2
3 #include "Core/AppData.hpp"
4 #include "Core/Controller.hpp"
5 #include "Core/Window.hpp"
6
7
11 class App {
12 private:
13     AppData _appData;
14     Window _window;
15     Controller _controller{_window};
16
21     void init(bool renderPreview = false);
22
23 public:
27     void run();
28 };
```

7.2 AppData.hpp

```
1 #pragma once
2
3 #include "AppData/AssetManager.hpp"
4 #include "AppData/ContextManager.hpp"
5 #include "AppData/ContextRepr.hpp"
6
7
11 class AppData {
12 private:
13     static AssetManager _assets;
14     static ContextManager _contexts;
15
16 public:
22     static const sf::Font& getFont(const std::string_view& name) { return _assets.getFont(name); }
28     static const sf::Texture& getTexture(const std::string_view& name) { return _assets.getTexture(name);
       }
34     static Context* findContext(const std::string_view& name) { return _contexts.find(name); }
35
39     void loadAssets() const { _assets.loadFromFiles(); }
43     void loadContexts() const { _contexts.loadFromFile(); }
47     void save() const { _contexts.save(); }
48 };
```

7.3 AssetManager.hpp

```
1 #pragma once
2
3 #include <map>
```

```
4 #include <SFML/Graphics.hpp>
5 #include <SFML/Audio.hpp>
6
7
11 class AssetManager {
12 private:
13     std::map<const std::string, const sf::Font, std::less<» _fonts;
19     void loadFont(const std::string& name, const std::string& fileName);
20
21     std::map<const std::string, const sf::Texture, std::less<» _textures;
27     void loadTexture(const std::string& name, const std::string& fileName);
28
29 public:
35     const sf::Font& getFont(const std::string_view& name);
41     const sf::Texture& getTexture(const std::string_view& name);
42
46     void loadFromFiles();
47 };
48
```

7.4 Context.hpp

```
1 #pragma once
2
3 #include <functional>
4 #include <any>
5 #include <string>
6
7
11 class Context {
12 public:
16     class ToStringConverter {
17         const std::function<std::string(const std::any&)> _func;
18     public:
23         explicit(false) ToStringConverter(const std::function<std::string(const std::any&)>& func) :
      _func{ func } {}
29         std::string operator()(const std::any& val) const { return _func(val); }
30     };
31
35     class Validator {
36     private:
37         std::function<bool(const std::any&)> _func;
38     public:
43         explicit(false) Validator(const std::function<bool(const std::any&)>& func = [](const std::any&)
      -> bool { return true; }) : _func{ func } {}
49         bool operator()(const std::any& data) const { return _func(data); }
50     };
51
52
53 private:
54     std::any _data;
55     const ToStringConverter _converter;
56     const Validator _validator;
57
58 public:
65     explicit Context(const std::any& data, const ToStringConverter& converter, const Validator&
      validator);
66
72     template <typename T>
73     T get() const { return std::any_cast<T>(_data); }
74
79     std::string string() const { return _converter(_data); }
80
88     template <typename T>
89     std::string string(const T& val) const { return _converter(val); }
90
96     bool set(const std::any& data);
102     bool set(std::any&& data);
103
109     bool validate(const std::any& potentialData) const { return _validator(potentialData); }
110 };
```

7.5 ContextManager.hpp

```
1 #pragma once
2
3 #include <map>
4 #include <SFML/Graphics.hpp>
5 #include "Context.hpp"
```

```
6
7
11 enum class PlatformControl {
12     Keyboard,
13     Mouse
14 };
15
16
20 class ContextManager {
21 private:
22     std::map<const std::string, Context, std::less<» _contexts;
30     void addContext(
31         const std::string& name,
32         const std::any& initialValue,
33         const Context::ToStringConverter& converter,
34         const Context::Validator& validator = {}
35     );
36
37 public:
41     ContextManager();
42
46     void loadFromFile();
50     void save();
51
52
58     Context* find(const std::string_view& name);
59 };
60
61
68 bool operator==(const sf::Event::KeyEvent& lhs, const sf::Event::KeyEvent& rhs);
```

7.6 ContextRepr.hpp

```
1 #pragma once
2
3 #include "ContextManager.hpp"
4
5
10 template <typename T>
11 class ContextRepr {
12 private:
13     T _dataRepr;
14     Context* const _context;
15
16 public:
17     ContextRepr() = delete;
22     explicit ContextRepr<T>(Context* const context);
28     ContextRepr<T>& operator=(const T& data);
34     ContextRepr<T>& operator=(const T&& data);
35
39     explicit (false) operator T() const { return _dataRepr; }
43     explicit operator std::string() const { return _context->string(); }
44
50     std::string string(const T& val) const { return _context->string(val); }
56     bool validate(const T& potentialData) const { return _context->validate(potentialData); }
57
61     void update() { _dataRepr = _context->get<T>(); }
62 };
63
64
66 // Definitions //
68
69 template<typename T>
70 inline ContextRepr<T>::ContextRepr(Context* const context) : _context{ context } {
71     if (_context == nullptr) {
72         throw std::invalid_argument{ "ContextRepr construction failed, because pointer to context is
       invalid.\n" };
73     }
74     update();
75 }
76
77 template<typename T>
78 inline ContextRepr<T>& ContextRepr<T>::operator=(const T& data) {
79     if (_context->set(data)) {
80         _dataRepr = data;
81     }
82     return *this;
83 }
84
85 template<typename T>
86 inline ContextRepr<T>& ContextRepr<T>::operator=(const T&& data) {
87     if (_context->set(data)) {
88         _dataRepr = std::move(data);
```

```
89      }
90      return *this;
91  }
```

7.7 Controller.hpp

```
1  #pragma once
2
3  #include "Controller/MenuNode.hpp"
4  #include "Window.hpp"
5
6
7  class App;
8
9
13 class Controller {
14 private:
15      MenuNode _root;
16
17      MenuNode* _current = nullptr;
18
19      void open();
20      void openLast();
21      void close();
22
23      Window& _window;
24      bool _preview = false;
25
26 public:
31      explicit Controller(Window& window);
32
36      void renderPreview();
40      void load();
41
46      bool isActive();
47
52      MenuBase* operator->() const;
53 };
```

7.8 MenuBase.hpp

```
1  #pragma once
2
3  #include <string>
4  #include <SFML/Graphics.hpp>
5
6  class Controller;
7
8
14 class MenuBase {
15 private:
16      friend Controller;
17      bool __isExiting__ = false;
18      bool __isClosing__ = false;
19      bool __openLast__ = false;
20      std::string __next__;
21
25      void __init__() {
26          __isClosing__ = false;
27          __openLast__ = false;
28          __next__.clear();
29      }
30
31 protected:
36      void open(const std::string_view& next) { __next__ = next; }
40      void openLast() { __openLast__ = true; }
44      void close() { __isClosing__ = true; }
48      void exit() { __isExiting__ = true; }
49
50 public:
55      virtual void handleEvent(const sf::Event& event) = 0;
59      virtual void update() = 0;
63      virtual void render() = 0;
64
68      virtual void init() = 0;
72      virtual void pause() = 0;
76      virtual void resume() = 0;
77
81      virtual ~MenuBase() = default;
82 };
```

7.9 MenuNode.hpp

```
1 #pragma once
2
3 #include <map>
4 #include <memory>
5 #include "MenuBase.hpp"
6
7
13 class MenuNode {
14 private:
15     std::unique_ptr<MenuBase> _item;
16     MenuNode* _parent = nullptr;
17     std::map<const std::string, const std::unique_ptr<MenuNode>, std::less<» _children;
18     MenuNode* _lastVisitedChild = nullptr;
19
20 public:
24     MenuNode() = default;
30     MenuNode(std::unique_ptr<MenuBase> item, MenuNode* parent = nullptr) : _item{ std::move(item) },
       _parent{ parent } {}
31
37     void addChild(const std::string& name, std::unique_ptr<MenuBase> child);
38
43     MenuBase* get() const { return _item.get(); }
48     MenuNode* getParent() const { return _parent; }
54     MenuNode* findChild(const std::string_view& name);
55
60     MenuNode* getLastVisitedChild() const { return _lastVisitedChild; }
65     void setLastVisitedChild(MenuNode* child) { _lastVisitedChild = child; }
66 };
```

7.10 PreView.hpp

```
1 #pragma once
2
3
7 class PreView {
8 public:
12     static void render();
13 };
```

7.11 Window.hpp

```
1 #pragma once
2
3 #include <SFML/Graphics.hpp>
4 #include <functional>
5
6
10 class Window {
11 private:
12     static sf::RenderWindow _window;
13     static std::function<sf::VideoMode()> _getVideoMode;
14
15     sf::String _title = "Platforms";
16     sf::ContextSettings _settings;
17     sf::Uint32 _style = sf::Style::Fullscreen;
18     sf::Clock _clock;
19
20 public:
25     static const sf::RenderWindow& window() { return _window; }
26
31     static sf::Vector2f getSize();
36     static sf::FloatRect getLocalBounds();
37
41     static void clear();
47     static void draw(const sf::Drawable& drawable, const sf::RenderStates& states =
       sf::RenderStates::Default);
51     static void display();
52
56     Window();
60     void open() const;
64     void close() const;
69     bool isOpen() const;
75     bool pollEvent(sf::Event& event) const;
81     void lockFPS(int FPS = 60);
82 };
```

7.12 Theme.hpp

```
1 #include <SFML/Graphics.hpp>
2
3
7 namespace theme {
8     extern const sf::Color Primary;
9     extern const sf::Color Secondary;
10     extern const sf::Color Tertiary;
11     extern const sf::Color Quaternary;
12     extern const sf::Color Gold;
13     extern const sf::Color Purple;
14     extern const sf::Color IndigoPurple;
15     extern const sf::Color IndigoPurpleShade;
16     extern const sf::Color NeonYellow;
17 }
```

7.13 Widget.hpp

```
1 #pragma once
2
3 #include <SFML/Graphics.hpp>
4 #include "../Menus/MenuItem.hpp"
5
6
10 class Widget : public MenuItem {
11 private:
12     sf::Vector2f _position;
13     sf::Vector2f _size;
14
15 protected:
20     void setSize(const sf::Vector2f& size) { _size = size; }
21
22 public:
28     Widget(const sf::Vector2f& position = { 0, 0 }, const sf::Vector2f& size = { 0, 0 }) : _position{
       position }, _size{ size } {}
32     Widget(const Widget& other) = default;
33
38     virtual void setPosition(const sf::Vector2f& position) { _position = position; }
43     const sf::Vector2f& getPosition() const { return _position; }
48     const sf::Vector2f& getSize() const { return _size; }
53     sf::FloatRect getLocalBounds() const { return sf::FloatRect{ _position.x, _position.y, _size.x,
       _size.y }; }
54
59     virtual void center(const sf::FloatRect& frame) { setPosition({ frame.left + frame.width / 2 -
       getSize().x / 2, frame.top + frame.height / 2 - getSize().y / 2 }); }
64     virtual void move(const sf::Vector2f& amount) { setPosition(_position + amount); }
65 };
66
```

7.14 Bar.hpp

```
1 #pragma once
2
3 #include <vector>
4 #include <string>
5 #include <functional>
6 #include <SFML/Graphics.hpp>
7 #include "../Widget.hpp"
8 #include "Text.hpp"
9 #include "../../Utilities/Math/Transitions.hpp"
10 #include "../../Utilities//Math/Transitionable.hpp"
11 #include "../../Core/AppData/ContextManager.hpp"
12 #include "../Theme.hpp"
13
14
18 class Emphasis : public sf::RectangleShape, public Transitionable {
19     void transition(const sf::Vector2f& amount) override { move(amount); }
20 };
21
22
27 template<typename T>
28 class Bar : public Widget {
29 private:
30     ContextRepr<T> _context;
31     Transitions::Bezier::Ease _transition{ &_emphasis };
32
33     std::vector<T> _contents;
34     std::vector<Text> _texts;
```

```
35      std::vector<sf::RectangleShape> _cells;
36      Emphasis _emphasis;
37
38      size_t _selected = 0;
39      size_t _activeCell = 0;
40
46      void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
47
48  public:
57      Bar(
58          float width,
59          const std::vector<T>& contents,
60          const sf::Font& font,
61          unsigned charaRcterSize,
62          const std::string_view& contextName
63      );
64
69      void setPosition(const sf::Vector2f& position) override;
70
75      void handleEvent(const sf::Event& event) override;
79      void update() override;
83      void init() override;
84  };
85
86
88  // Definitions //
90
91  template<typename T>
92  void Bar<T>::draw(sf::RenderTarget& target, sf::RenderStates states) const {
93      for (const auto& text : _texts)
94          target.draw(text);
95
96      target.draw(_emphasis);
97
98      for (const auto& cell : _cells)
99          target.draw(cell);
100 }
101
102
103 template<typename T>
104 Bar<T>::Bar(
105     float width,
106     const std::vector<T>& contents,
107     const sf::Font& font,
108     unsigned charaRcterSize,
109     const std::string_view& contextName
110 ) :
111     _context{ AppData::findContext(contextName) },
112     _contents{ contents }
113 {
114     sf::Vector2f cellSize = { width / static_cast<float>(_contents.size()), Text{ "0", font,
      charaRcterSize }.getSize().y * 2.f };
115
116     _cells.resize(contents.size());
117     for (size_t i = 0; i < _cells.size(); i++) {
118         _cells[i].setSize(cellSize);
119         _cells[i].move({ cellSize.x * static_cast<float>(i), 0.f });
120         _cells[i].setFillColor(sf::Color::Transparent);
121         _cells[i].setOutlineColor(theme::IndigoPurple);
122         _cells[i].setOutlineThickness(2.f);
123     }
124
125     _texts.resize(contents.size());
126     for (size_t i = 0; i < _texts.size(); i++) {
127         _texts[i] = Text{ _context.string(contents[i]), font, charaRcterSize };
128         _texts[i].center(_cells[i].getLocalBounds());
129         _texts[i].move(_cells[i].getPosition());
130         _texts[i].setFillColor(theme::IndigoPurple);
131     }
132
133     _emphasis.setSize(cellSize);
134     _emphasis.setPosition(_cells[_selected].getPosition());
135     _emphasis.setFillColor(theme::Tertiary);
136
137     setSize({ cellSize.x * static_cast<float>(_contents.size()), cellSize.y });
138
139     addTransition(&_transition);
140 }
141
142
143 template<typename T>
144 void Bar<T>::setPosition(const sf::Vector2f& position) {
145     for (sf::RectangleShape& cell : _cells) {
146         cell.setPosition(position + (cell.getPosition() - getPosition()));
147     }
148     for (Text& content : _texts) {
149         content.setPosition(position + (content.getPosition() - getPosition()));
```

```
150     }
151
152     _emphasis.setPosition(position + (_emphasis.getPosition() - getPosition()));
153
154     Widget::setPosition(position);
155 }
156
157
158 template<typename T>
159 void Bar<T>::handleEvent(const sf::Event& event) {
160     if (event.type == sf::Event::MouseButtonPressed) {
161         for (size_t i = 0; i < _cells.size(); i++) {
162             if (_cells[i].getGlobalBounds().contains(sf::Vector2f{
      sf::Mouse::getPosition(Window::window()) })) {
163                 _context = _contents[i];
164                 _selected = i;
165             }
166         }
167     }
168 }
169
170 template<typename T>
171 void Bar<T>::update() {
172     if (_context != _contents[_activeCell] && !_transition.isActive()) {
173         _transition.start(_cells[_selected].getPosition() - _emphasis.getPosition(), 200);
174         _activeCell = _selected;
175     }
176
177     MenuItem::update();
178 }
179
180 template<typename T>
181 void Bar<T>::init() {
182     MenuItem::init();
183
184     _context.update();
185     for (size_t i = 0; i < _texts.size(); i++) {
186         if (_contents[i] == _context) {
187             _emphasis.setPosition(_cells[i].getPosition());
188             _selected = i;
189             _activeCell = i;
190         }
191     }
192 }
```

7.15 Button.hpp

```
1 #pragma once
2
3 #include <functional>
4 #include "../Widget.hpp"
5 #include "Text.hpp"
6
7
11 class Button : public Widget {
12 private:
13     Text _text;
14     std::function<void()> _callback;
15
21     bool isInside(const sf::Vector2f& point) const;
25     void triggerCallback() const;
26
32     void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
33
34 public:
42     Button(
43         const sf::String& text,
44         const sf::Font& fontStyle,
45         unsigned characterSize,
46         const std::function<void()>& callback = nullptr
47     );
48
53     void setPosition(const sf::Vector2f& position) override { Widget::setPosition(position);
      _text.center(getLocalBounds()); }
54
59     void handleEvent(const sf::Event& event) override;
63     void resume() override;
64 };
```

7.16 InputField.hpp

```
1 #pragma once
2
3 #include <functional>
4 #include <string>
5 #include "../Widget.hpp"
6 #include "Text.hpp"
7
8
12 class InputField : public Widget {
13 private:
14     ContextRepr<sf::Event::KeyEvent> _context;
15
16     Text _text;
17     std::string _string;
18     sf::RectangleShape _frame;
19
20     sf::Clock _clock;
21     void setActive(bool isActive);
22     std::string _activeString;
23     bool _isActive = false;
24
30     void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
31
32 public:
39     InputField(
40         const sf::Font& fontStyle,
41         unsigned characterSize,
42         const std::string_view& contextName
43     );
44
49     void setPosition(const sf::Vector2f& position) override;
50
55     void handleEvent(const sf::Event& event) override;
59     void update() override;
63     void init() override;
64 };
```

7.17 Text.hpp

```
1 #pragma once
2
3 #include "../Widget.hpp"
4
5
9 class Text : public Widget {
10 private:
11     sf::Text _text;
12
18     void draw(sf::RenderTarget& target, sf::RenderStates states) const override { target.draw(_text); }
19
20 public:
24     Text() = default;
31     Text(const sf::String& text, const sf::Font& fontStyle, unsigned characterSize);
35     Text(const Text&) = default;
36
41     void setPosition(const sf::Vector2f& position) override { Widget::setPosition(position);
       _text.setPosition(position + getSize() / 2.f); }
46     void setFillColor(const sf::Color& color) { _text.setFillColor(color); }
51     void setString(const sf::String& string);
56     const sf::String& getString() const { return _text.getString(); }
57 };
```

7.18 GameMenu.hpp

```
1 #pragma once
2
3 #include "Menu.hpp"
4 #include "GameMenu/Track.hpp"
5 #include "GameMenu/Player.hpp"
6 #include "GameMenu/PauseScreen.hpp"
7
8
12 class GameMenu : public Menu {
13 private:
14     Track _track;
15     Player _player;
16     sf::CircleShape _playerCircle;
```

```
17      PauseScreen _pauseScreen{ [this]() {resume(); }, [this]() {close(); pause(); }, [this]() {init();} };
18
19      bool _gameOver = false;
20
21  public:
25      GameMenu();
26
30      void handleEvent(const sf::Event& event) override;
34      void update() override;
38      void render() override;
42      void init() override;
46      void pause() override;
50      void resume() override;
51  };
```

7.19 PauseScreen.hpp

```
1  #include "../MenuItem.hpp"
2
3  #include "../../GUI/Widgets/Text.hpp"
4  #include "../../GUI/Widgets/Button.hpp"
5
6
10 class PauseScreen : public MenuItem {
11 private:
12      sf::RectangleShape _bg{Window::getSize()};
13      Text _text{ "Resume", AppData::getFont("Dameron"), 84u };
14      Button _backButton;
15      Button _playAgainButton;
16
17      const std::function<void()> _resumeMenu;
18
19      bool _gameOver = false;
20
26      void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
27
28 public:
35      explicit PauseScreen(const std::function<void()>& resumeMenu, const std::function<void()>& closeMenu,
        const std::function<void()>& initMenu);
36
40      void gameOver();
45      void handleEvent(const sf::Event& event) override;
49      void init() override;
50 };
```

7.20 Platform.hpp

```
1  #pragma once
2
3  #include <SFML/Graphics.hpp>
4  #include "PreCalculator.hpp"
5  #include "../../Utilities/Math/Angle.hpp"
6
7
11 class Platform : public sf::Drawable {
12 private:
13      static const float _initInnerRadius;
14      static const float _initOuterRadius;
15
16      static sf::Vector2f _origin;
17      static float _maxRadius;
18
19      // Non-static
20      const PreCalculator& _preCalc;
21
22      float _scalingRatio;
27      void setScale(unsigned speed);
28
29      float _innerRadius = _initInnerRadius;
30      float _outerRadius = _initOuterRadius;
31      float _rotation = 0_deg;
32      const float _width;
33
39      void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
40
41 public:
46      static void setOrigin(const sf::Vector2f& origin);
47
55      Platform(const PreCalculator& preCalc, float rotation, float width, unsigned speed);
```

```
56
61      float getInnerRadius() const { return _innerRadius; }
66      float getOuterRadius() const { return _outerRadius; }
71      float getRotation() const { return _rotation; }
76      float getWidth() const { return _width; }
77
83      bool isInside(const PolarVector& point) const;
84
89      void rotate(float angle);
90
94      void update();
99      bool isDead() const;
100 };
```

7.21 PlatformContainer.hpp

```
1  #pragma once
2
3  #include <deque>
4  #include <random>
5  #include "Platform.hpp"
6  #include "../../Utilities/Math/Transitionable.hpp"
7
8
12 class PlatformContainer : public sf::Drawable, public Transitionable {
13 private:
14      const PreCalculator& _preCalc;
15      std::deque<Platform> _platforms;
16      float _platformWidth = 360_deg / 8;
17
18      unsigned _counter = 0;
19      unsigned _scaleSpeed = 20;  //the lower the faster
20
21      float _rotation = 0_deg;
22
23      // Random generation
24      std::mt19937 _randomEngine{ std::random_device{}() };
31      int generateRandom(int from, int to);
32      bool _random = true;
33
39      void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
40
41 public:
46      explicit PlatformContainer(const PreCalculator& preCalc);
47
52      float getPlatformWidth() const { return _platformWidth; }
53
59      bool isInside(const PolarVector& point) const;
67      bool AI_help(PolarVector playerFeet, int& switchingState);
68
73      void rotate(float angle);
78      void transition(const sf::Vector2f& amount) override { rotate(amount.x); }
79
83      void update();
89      void init(unsigned laneCount, unsigned speed);
90 };
91
```

7.22 Player.hpp

```
1  #pragma once
2
3  #include "../MenuItem.hpp"
4  #include "../../Utilities/Math/Transitions.hpp"
5  #include "../../Utilities/Math/Transitionable.hpp"
6
7
11 class PlayerSprite : public sf::Sprite, public Transitionable {
12 private:
13      sf::Vector2f _initPos;
14
15 public:
19      PlayerSprite();
20
25      void setInitPos(const sf::Vector2f& pos) { _initPos = pos; setPosition(pos); }
30      const sf::Vector2f& getInitPos() const { return _initPos; }
31
36      void transition(const sf::Vector2f& amount) override { move(amount); }
37
```

```
41     void init() { setPosition(_initPos); }
42 };
43
44
48 class Player : public MenuItem {
49 private:
50     PlayerSprite _sprite;
51     const sf::Vector2f _offset = { 0.f, 30.f };
52
53     bool _jumpKeyPressed = false;
54     bool _resetJump = false;
55
56     // Transition(s)
57     Transitions::Jump _transition{ &_sprite };
58
59     // Context accessor(s)
60     ContextRepr<sf::Event::KeyEvent> _jumpKey{ AppData::findContext("jumpKey") };
61
62 protected:
66     void startJump();
72     void draw(sf::RenderTarget& target, sf::RenderStates states) const override { target.draw(_sprite); }
73
74 public:
78     Player();
79
84     bool isJumping() const { return _transition.isActive(); }
89     sf::Vector2f getFeet() const { return _sprite.getInitPos() + _offset; }
94     void setPosition(const sf::Vector2f& feet);
95
100     void handleEvent(const sf::Event& event) override;
104     void update() override;
108     void init() override;
112     void pause() override;
113 };
```

7.23 PreCalculator.hpp

```
1 #pragma once
2
3 #include <map>
4 #include <SFML/Graphics.hpp>
5 #include "../../Utilities/Math/PolarVector.hpp"
6
7
11 class PreCalculator {
12 private:
13     struct cmpVector2i { bool operator()(const sf::Vector2i& lhs, const sf::Vector2i& rhs) const; };
14     std::map<sf::Vector2i, PolarVector, cmpVector2i> _polarVectorMap;
15
16 public:
20     PreCalculator();
21
27     const PolarVector& getPolarVector(const sf::Vector2f& vector) const {
28         return _polarVectorMap.at(sf::Vector2i{ vector });
29     }
30 };
```

7.24 Track.hpp

```
1 #pragma once
2
3 #include "../../Menus/MenuItem.hpp"
4 #include "PreCalculator.hpp"
5 #include "../../Utilities/Math/Transitions.hpp"
6 #include "PlatformContainer.hpp"
7
8
12 class Track : public MenuItem {
13 private:
14     // "Observables"
15     PreCalculator _preCalc;
16     Transitions::EaseInOut _transition{ &_platforms };
17
18     // Managed items
19     PlatformContainer _platforms{ _preCalc };
20     sf::VertexArray _shader{ sf::TriangleFan };
21
22     // Variables
23     sf::Vector2f _mouse{ sf::Vector2f{sf::Mouse::getPosition(Window::window())} };
```

```
24      bool _isDragged = false;
25      int _switchingState = 0;
26      bool _switchingLeft = false;
27      bool _switchingRight = false;
28
33      void switchLane(float direction);
37      void switchLanes();
38
39      // Contexts
40      unsigned _laneCount = 8u;
41      ContextRepr<unsigned> _platformSpeed{ AppData::findContext("speed") };
42      ContextRepr<PlatformControl> _platformControl{ AppData::findContext("platformControl") };
43      ContextRepr<sf::Event::KeyEvent> _switchKey1{ AppData::findContext("switchKey1") };
44      ContextRepr<sf::Event::KeyEvent> _switchKey2{ AppData::findContext("switchKey2") };
45      ContextRepr<bool> _holdSwitch{ AppData::findContext("holdSwitch") };
46
47  protected:
53      void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
54
59      void handleMouseEvent(const sf::Event& mouseEvent);
64      void handleKeyEvent(const sf::Event& keyEvent);
65
66  public:
70      Track();
71
77      bool isOnPlatforms(const sf::Vector2f& point) const;
83      bool AI_jump(const sf::Vector2f& playerFeet);
84
89      void handleEvent(const sf::Event& event) override;
93      void update() override;
97      void init() override;
101      void resume() override;
102 };
```

7.25 MainMenu.hpp

```
1  #pragma once
2
3  #include "Menu.hpp"
4  #include "GameMenu/Track.hpp"
5  #include "GameMenu/Player.hpp"
6
7
11 class TrackAI : public Track {
12 public:
17      void handleEvent(const sf::Event&) override { /* AI only */ }
18 };
22 class PlayerAI : public Player {
23 public:
27      void jump() { startJump(); }
32      void handleEvent(const sf::Event&) override { /* AI only */ }
33 };
37 class Shader : public MenuItem {
38      sf::RectangleShape _item{ Window::getSize() };
44      void draw(sf::RenderTarget& target, sf::RenderStates states) const override { target.draw(_item); }
45 public:
49      Shader() { _item.setFillColor({ 0, 0, 0, 220 }); }
50 };
51
52
56 class MainMenu : public Menu {
57 private:
58      // Background
59      TrackAI* _track;
60      PlayerAI* _player;
61
62 public:
66      MainMenu();
67
72      void handleEvent(const sf::Event& event) override;
76      void update() override;
77 };
```

7.26 Menu.hpp

```
1  #pragma once
2
3  #include <memory>
4  #include "../Utilities/STL/vector.hpp"
```

```
5 #include "../Core/Controller/MenuBase.hpp"
6 #include "../Core/AppData.hpp"
7 #include "../Core/Window.hpp"
8 #include "MenuItem.hpp"
9
10
14 class Menu : public MenuBase {
15 private:
16     util::vector<std::unique_ptr<MenuItem>> _items;
17
18     bool _isPaused = false;
19
20 protected:
25     void addMenuItem(std::unique_ptr<MenuItem> item);
26
31     bool isPaused() const { return _isPaused; }
32
33 public:
38     void handleEvent(const sf::Event& event) override;
42     void update() override;
46     void render() override;
47
51     void init() override;
55     void pause() override;
59     void resume() override;
60 };
```

7.27 MenuItem.hpp

```
1 #pragma once
2
3 #include <vector>
4 #include <SFML/Graphics.hpp>
5 #include "../Core/AppData.hpp"
6 #include "../Core/Window.hpp"
7 #include "../Utilities/Math/Transitions.hpp"
8
9
13 class MenuItem : public sf::Drawable {
14 private:
15     bool _isPaused = false;
16
17     std::vector<Transition*> _transitions;
18
19 protected:
24     bool isPaused() const { return _isPaused; }
25
30     void addTransition(Transition* transition) { _transitions.push_back(transition); }
31
32 public:
36     MenuItem() = default;
41     explicit MenuItem(const std::vector<Transition*>& transitions) : _transitions{transitions} {}
42
46     virtual void handleEvent(const sf::Event& event) { /*not pure*/ }
50     virtual void update();
54     virtual void init();
58     virtual void pause();
62     virtual void resume();
63 };
64
```

7.28 OptionsMenu.hpp

```
1 #include "Menu.hpp"
2
3
7 class OptionsMenu : public Menu {
8 public:
12     OptionsMenu();
13
18     void handleEvent(const sf::Event& event) override;
19 };
```

7.29 Math.hpp

```
1 #pragma once
```

```
2
3 #include <cmath>
4 #include <SFML/Graphics.hpp>
5 #include "Math/Angle.hpp"
6
7
11 namespace math {
18     template <typename T>
19     T square(T num) {
20         return num * num;
21     }
27     inline float squaref(float num) {
28         return num * num;
29     }
30
37     float calcDistance(const sf::Vector2f& a, const sf::Vector2f& b);
38
44     float calcAngle(const sf::Vector2f& position);
45
53     bool isBetween(float val, float smaller, float bigger);
54
63     std::vector<sf::Vector2f> getArcPoints(float angle, float spread, float radius, int maxpts);
64 }
```

7.30 Angle.hpp

```
1 #pragma once
2
3
4 namespace math {
5     extern const float PI;
6 }
7
8
12 float operator"" _deg(long double degree);
16 float operator"" _deg(unsigned long long degree);
17
18 namespace math {
24     float convertToDeg(float radian);
25 }
```

7.31 BezierEasing.hpp

```
1 #pragma once
2
3 #include <vector>
4 #include <SFML/Graphics.hpp>
5
6
10 class BezierEasing {
11 public:
17     BezierEasing(const sf::Vector2f& p1, const sf::Vector2f& p2);
18
24     float GetEasingProgress(float t);
25
26 private:
27     // Calculate support vectors
28     float VecACoord(float p1_coord, float p2_coord) const;
29     float VecBCoord(float p1_coord, float p2_coord) const;
30     float VecCCoord(float p1_coord) const;
31
32     //Calculate the Bezier point
33     float CalcBezier(float time, float p1_coord, float p2_coord) const;
34
35     //Calculate Bezier Slope
36     float GetSlope(float time, float p1_coord, float p2_coord) const;
37
38     float BinarySubdivide(float time, float interval_start, float next_interval_step, float p1_coord,
       float p2_coord) const;
39
40     float NewtonRaphsonIterate(float time, float guessed_t, float p1_coord, float p2_coord) const;
41
42     float GetXForTime(float time);
43
44     //Check if the points are in the valid range
45     bool CheckPoints() const;
46
47
48 // Variables
```

```
49      sf::Vector2f _p1{ 0.f, 0.f };
50      sf::Vector2f _p2{ 0.f, 0.f };
51
52      std::vector<float> _sample_values;
53
54      float _last_value = 0.f;
55
56      bool _valid = true;
57  };
```

7.32 PolarVector.hpp

```
1  #pragma once
2
3
7  struct PolarVector {
8      float radius;
9      float angle;
10      PolarVector(float radius_ = 0, float angle_ = 0) : radius{ radius_ }, angle{ angle_ } {}
11 };
```

7.33 Transitionable.hpp

```
1  #pragma once
2
3
7  class Transitionable {
8  public:
9      virtual void transition(const sf::Vector2f& amount) = 0;
10 };
```

7.34 Transitions.hpp

```
1  #pragma once
2
3  #include <functional>
4  #include <SFML/System.hpp>
5  #include <SFML/Graphics.hpp>
6  #include "Transitionable.hpp"
7  #include "BezierEasing.hpp"
8  #include "../Math.hpp"
9
10
14 class Transition {
15 private:
16     Transitionable* _object = nullptr;
17     const std::function<sf::Vector2f(int elapsedTime)> _getProgression;
18     std::function<sf::Vector2f()> _correctDistance;
19
20     int _time = 0;  // in milliseconds
21     sf::Vector2f _distance = { 0, 0 };
22
23     bool _isActive = false;
24
25     // During transition variables
26     sf::Clock _clock;
27     int _elapsedTime = 0;
28     sf::Vector2f _distanceTraveled = { 0.f, 0.f };
29
30     // Paused
31     bool _isPaused = false;
32     int _pausedTime = 0;
33     sf::Clock _pausedClock;
34
35 protected:
42     explicit Transition(
43         Transitionable* object,
44         const std::function<sf::Vector2f(int elapsedTime)>& getProgression,
45         const std::function<sf::Vector2f()>& correctDistance = nullptr
46     );
47
52     int getDurationTime() const { return _time; }
57     const sf::Vector2f& getDurationDistance() const { return _distance; }
62     int getElapsedTime() const { return _elapsedTime; }
67     const sf::Vector2f& getDistanceTraveled() const { return _distanceTraveled; }
```

```
68
69 public:
74     bool isActive() const { return _isActive; }
75
82     virtual bool start(const sf::Vector2f& distance, int time) {
83         if (!_isActive) {
84             init();
85             _distance = distance;
86             _elapsedTime = 0;
87             _distanceTraveled = { 0.f, 0.f };
88             _isPaused = false;
89             _pausedTime = 0;
90             _time = time;
91             _clock.restart();
92             _isActive = true;
93             return true;
94         }
95         return false;
96     }
100     void update();
104     virtual void init() { _isActive = false; }
108     virtual void pause() { _isPaused = true; _pausedClock.restart(); }
112     virtual void resume() { _isPaused = false; _pausedTime +=
        _pausedClock.getElapsedTime().asMilliseconds(); }
113
117     virtual ~Transition() = default;
118 };
119
123 namespace Transitions {
127     class EaseInOut : public Transition {
128     private:
129         sf::Vector2f _acc;  // max acceleration during transition
130         sf::Vector2f _velocity2;
131
132         sf::Vector2f calcAcc(const sf::Vector2f& distance, int time) const { return distance /
        static_cast<float>(math::square(time / 2)); }
133         sf::Vector2f calcVelocity2(const sf::Vector2f& acc, int time) const { return acc *
        static_cast<float>(time / 2); }
134
135     public:
140         explicit EaseInOut(Transitionable* object);
141
148         bool start(const sf::Vector2f& distance, int time) override {
149             if (!isActive()) {
150                 _acc = calcAcc(distance, time);
151                 _velocity2 = calcVelocity2(_acc, time);
152                 return Transition::start(distance, time);
153             }
154             return false;
155         }
156     };
157
161     namespace Bezier {
165         class Ease : public Transition {
166         private:
167             BezierEasing _bezier{ { 0.25f, 0.1f }, { 0.25f, 1.f } };
168
169         public:
174             explicit Ease(Transitionable* object);
175         };
176     }
177
181     class Jump : public Transition {
182     private:
183         sf::Vector2f _acc;
184         sf::Vector2f _velocity;
185
192         sf::Vector2f calcAcc(const sf::Vector2f& distance, int time) const {
193             return - 2.f * distance / math::squaref(static_cast<float>(time) / 2);
194         }
201         sf::Vector2f calcVelocity(const sf::Vector2f& acc, int time) const {
202             return - acc / 2.f * static_cast<float>(time);
203         }
204
205     public:
210         explicit Jump(Transitionable* object);
211
218         bool start(const sf::Vector2f& distance, int time) override {
219             if (!isActive()) {
220                 _acc = calcAcc(distance, time);
221                 _velocity = calcVelocity(_acc, time);
222                 return Transition::start(distance, time);
223             }
224             return false;
225         }
226     };
227 }
```

7.35 vector.hpp

```
1 #pragma once
2
3 #include <utility>
4
5
6 namespace util {
11      template <typename T>
12      class vector {
13      public:
14          class iterator;
15
16      private:
17          T* _ptr = nullptr;
18          std::size_t _size = 0;
19
20      public:
24          vector() = default;
25          vector(const vector&) = delete;
26          vector& operator=(const vector&) = delete;
27
33          template <typename... Args>
34          void emplace_back(Args &&...args) {
35              auto* tmp = new T[_size + 1u];
36              for (std::size_t i = 0; i < _size; i++) {
37                  tmp[i] = std::move(_ptr[i]);
38              }
39              tmp[_size++] = T{ std::forward<Args>(args)... };
40              delete[] _ptr;
41              _ptr = tmp;
42          }
43
48          iterator begin() { return iterator(_ptr, 0, _size); }
53          iterator end() { return iterator(_ptr, _size, _size); }
54
55          /*
56           * @brief   Deletes the contents of the class
57          */
58          ~vector() {
59              delete[] _ptr;
60          }
61
65          class iterator {
66          private:
67              T* _vector;
68              std::size_t _index = 0;
69              std::size_t _size;
70
71          public:
78              explicit iterator(T* vector, std::size_t index, std::size_t size) : _vector{ vector },
      _index{ index }, _size{ size } {}
83              iterator& operator++() {
84                  ++_index;
85                  return *this;
86              }
92              iterator operator++(int) {
93                  iterator retval = *this;
94                  ++(*this);
95                  return retval;
96              }
102             bool operator==(const iterator& other) const { return _index == other._index; }
107             T& operator*() { return _vector[_index]; }
108         };
109     };
110 }
```