Doodle Jump

Generated by Doxygen 1.9.2

Chapter 1

Advanced-Programming-DoodleJump

Doodle Jump game implemented with SFML for the Advanced Programming course at the University of Antwerp

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Namespace Documentation

6.1 Model Namespace Reference

Namespace holds all Models.

Classes

- class AbstractFactory
- class Background
- class Bonus
- class CollisionBox
- class Entity

    Class for Entity object.
- class HorizontalPlatform
- class Jetpack
- class Platform
- class Player

    Class for Player object, derives from Entity.
- class Score
- class Spring
- class StaticPlatform
- class TemporaryPlatform
- class VerticalPlatform

Enumerations

- enum Type {
  ePlayer = 0 , **eBonus** = 1 , **eStatic** = 2 , **eHorizontal** = 3 ,
  eVertical = 4 , **eTemporary** = 5 , **eBackground** = 6 , **eJetpack** = 7 ,
  eSpring = 8 , **eScore** = 9 }

6.1.1 Detailed Description

Namespace holds all Models.

Namespace holds all model.

6.1.2 Enumeration Type Documentation

6.1.2.1 Type

```
enum Model::Type
```

@Brief Enum containing enumerated Entities

6.2 Observer Namespace Reference

Namespace holds Observer pattern.

Classes

- class Observer

  Class for Observer of Observer pattern.
- class Subject

  Class for Subject / Observable of observer pattern.

6.2.1 Detailed Description

Namespace holds Observer pattern.

Namespace holds observer pattern.

6.3 Settings Namespace Reference

Namespace for Settings used in World.

Enumerations

- enum Difficulty {
  eEasy = 0 , **eNormal** , **eDifficult** , **eHard** ,
  eExtreme }

  Enum holding different difficulties.

6.3.1 Detailed Description

Namespace for Settings used in World.

6.3.2 Enumeration Type Documentation

6.3.2.1 Difficulty

`enum Settings::Difficulty`

Enum holding different difficulties.

Difficulty

6.4 Utils Namespace Reference

Namespace holds all Utilities.

Classes

- class Camera

  Class for Camera.
- class Exception
- class FileException
- class Random

  Class for Random.
- struct Resourceholder
- class Resourcemanager
- class Stopwatch

  Class for Stopwatch.
- class Utilities

  Class for Utilities.

6.4.1 Detailed Description

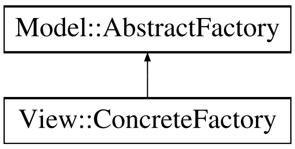
Namespace holds all Utilities.

Chapter 7

Class Documentation

7.1 Model::AbstractFactory Class Reference

Inheritance diagram for Model::AbstractFactory:



Public Member Functions

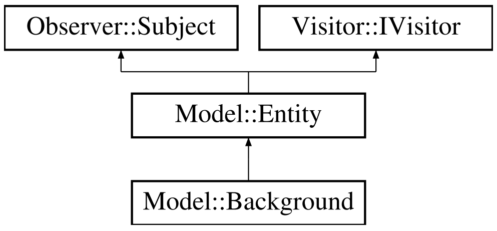
- virtual std::shared_ptr< Model::Player > **createPlayer** ()=0
- virtual std::shared_ptr< Model::Entity > **createStaticPlatform** ()=0
- virtual std::shared_ptr< Model::Entity > **createHorizontalPlatform** ()=0
- virtual std::shared_ptr< Model::Entity > **createVerticalPlatform** ()=0
- virtual std::shared_ptr< Model::Entity > **createTemporaryPlatform** ()=0
- virtual std::shared_ptr< Model::Entity > **createSpring** ()=0
- virtual std::shared_ptr< Model::Entity > **createJetpack** ()=0
- virtual std::shared_ptr< Model::Entity > **createBackground** ()=0
- virtual std::shared_ptr< Model::Score > **createScore** ()=0

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Abstract↵ Factory.h

7.2 Model::Background Class Reference

Inheritance diagram for Model::Background:



Public Member Functions

- Model::Type getType () const override

  Get type of Entity object.
- void move (bool collision) override

  move Entity object

Additional Inherited Members

7.2.1 Member Function Documentation

7.2.1.1 getType()

```
Model::Type Model::Background::getType ( ) const  [inline], [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.2.1.2 move()

```
void Model::Background::move (
             bool collision )  [inline], [override], [virtual]
```

move Entity object

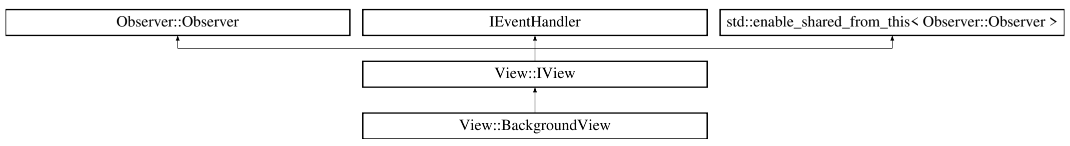
Implements Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Background.↩
  h

7.3 View::BackgroundView Class Reference

Inheritance diagram for View::BackgroundView:



Public Member Functions

- **BackgroundView** (const std::shared_ptr< Model::Entity > &entity, const std::shared_ptr< sf::Render←Window > &window)
- void handleEvent (const DrawEvent &event) override
- void handleEvent (const OutOfViewEvent &event) override

Additional Inherited Members

7.3.1 Member Function Documentation

7.3.1.1 handleEvent() `[1/2]`

```
void BackgroundView::handleEvent (
            const DrawEvent & event )  [override], [virtual]
```

Reimplemented from IEventHandler.

7.3.1.2 handleEvent() `[2/2]`

```
void BackgroundView::handleEvent (
            const OutOfViewEvent & event )  [override], [virtual]
```

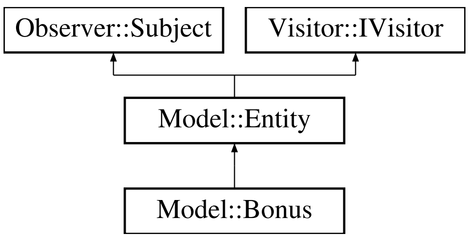
Reimplemented from IEventHandler.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/view/Background←View.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/src/view/Background←View.cpp

7.4 Model::Bonus Class Reference

Inheritance diagram for Model::Bonus:



Public Types

- enum Sort { **eJetpack** = 0 , **eSpring** = 1 }

  Enum containing enumerated Bonuses.

Public Member Functions

- Model::Type getType () const override

  Get type of Entity object.
- Sort getMSort () const

  Get sort of Bonus object.
- void setMSort (Sort sort)

  Set sort of Bonus object.

Additional Inherited Members

7.4.1 Member Function Documentation

7.4.1.1 getMSort()

```
Bonus::Sort Bonus::getMSort ( ) const
```

Get sort of Bonus object.

Returns

7.4.1.2 getType()

```
Model::Type Model::Bonus::getType ( ) const  [inline], [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.4.1.3 setMSort()

```
void Bonus::setMSort (
            Bonus::Sort sort )
```

Set sort of Bonus object.

Parameters

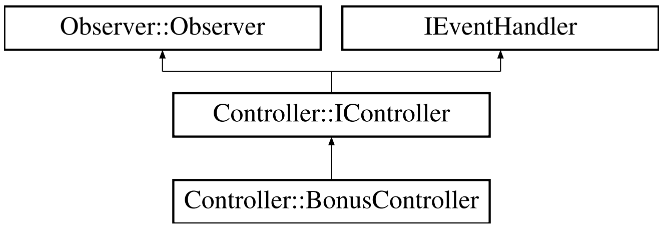
sort	Bonus::Sort

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Bonus.↩ h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Bonus.cpp

7.5 Controller::BonusController Class Reference

Inheritance diagram for Controller::BonusController:



Public Member Functions

- **BonusController** (std::shared_ptr< Model::Entity > &entity)

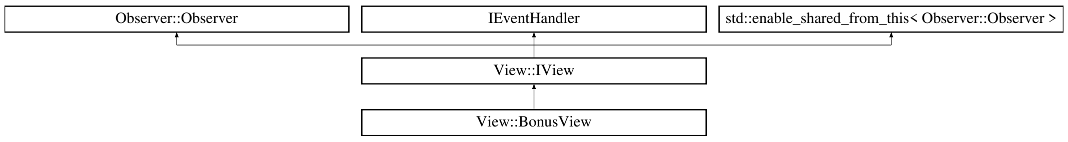
Additional Inherited Members

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/controller/Bonus↩ Controller.h

7.6 View::BonusView Class Reference

Inheritance diagram for View::BonusView:



Public Member Functions

- **BonusView** (const std::shared_ptr< Model::Entity > &entity, const std::shared_ptr< sf::RenderWindow > &window)

Additional Inherited Members

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

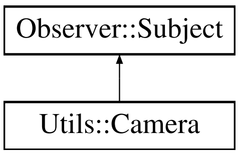
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/view/Bonus↩ View.h

7.7 Utils::Camera Class Reference

Class for Camera.

```
#include <Camera.h>
```

Inheritance diagram for Utils::Camera:



Public Member Functions

- virtual ∼**Camera** ()=default

  Default destructor.
- **Camera** (const Camera &)=delete

  Deleted copy constructor.
- Camera & operator= (const Camera &)=delete

  Deleted assignment operator.
- void **reset** ()
- std::pair< float, float > getWorldDimensions () const

  Get world dimensions.
- void setWorldDimensions (float right, float top, float left=0.f, float bottom=0.f)

  Set world dimensions.
- std::pair< float, float > getWindowDimensions () const

  Get window dimensions.
- void setWindowDimensions (float right, float bottom, float left=0.f, float top=0.f)

  Set viewport / window dimensions.
- std::pair< float, float > transform (float x, float y, float left=0.f, float top=0.f) const

  Transforms world coordinates to viewport / window coordinates.
- std::pair< float, float > inverseTransform (float x, float y) const

  Transforms viewport / window coordinates to world coordinates.
- void move (float x, float y)

Move Camera.
- float getX () const

  Get x coordinate of Camera.
- float getY () const

  Get y coordinate of Camera.
- float getMaxHeight () const

  Get maximum height of Camera.
- float **getLastMaxHeight** () const

  Get last maximum height of Camera.
- bool isMaxHeight (float height)

  Check if given height is greater or equal to current maximum height.

Static Public Member Functions

- static Camera & getInstance ()

  Get instance of Camera object.

7.7.1 Detailed Description

Class for Camera.

7.7.2 Member Function Documentation

7.7.2.1 getInstance()

```
Camera & Camera::getInstance ( )  [static]
```

Get instance of Camera object.

Returns

Camera

7.7.2.2 getMaxHeight()

```
float Utils::Camera::getMaxHeight ( ) const  [inline]
```

Get maximum height of Camera.

Returns

float

7.7.2.3 getWindowDimensions()

```
std::pair< float, float > Camera::getWindowDimensions ( ) const
```

Get window dimensions.

Returns

7.7.2.4 getWorldDimensions()

```
std::pair< float, float > Camera::getWorldDimensions ( ) const
```

Get world dimensions.

Returns

std::pair<float, float>

7.7.2.5 getX()

```
float Utils::Camera::getX ( ) const  [inline]
```

Get x coordinate of Camera.

Returns

float

7.7.2.6 getY()

```
float Utils::Camera::getY ( ) const  [inline]
```

Get y coordinate of Camera.

Returns

float

7.7.2.7 inverseTransform()

```
std::pair< float, float > Camera::inverseTransform (
             float x,
             float y ) const
```

Transforms viewport / window coordinates to world coordinates.

Parameters

x	float
y	float

Returns

std::pair<float, float>

7.7.2.8 isMaxHeight()

```
bool Camera::isMaxHeight (
             float height )
```

Check if given height is greater or equal to current maximum height.

Parameters

height	float

Returns

bool

7.7.2.9 move()

```
void Camera::move (
             float x,
             float y )
```

Move Camera.

Parameters

x	float
y	float

7.7.2.10 operator=()

```
Camera & Utils::Camera::operator= (
             const Camera & )  [delete]
```

Deleted assignment operator.

Returns

Camera

7.7.2.11 setWindowDimensions()

```
void Camera::setWindowDimensions (
             float right,
             float bottom,
             float left = 0.f,
             float top = 0.f )
```

Set viewport / window dimensions.

Parameters

right	float
bottom	float
left	float
top	float

7.7.2.12 setWorldDimensions()

```
void Camera::setWorldDimensions (
             float right,
             float top,
             float left = 0.f,
             float bottom = 0.f )
```

Set world dimensions.

Parameters

right	float
top	float
left	float
bottom	float

7.7.2.13 transform()

```
std::pair< float, float > Camera::transform (
             float x,
             float y,
             float left = 0.f,
             float top = 0.f ) const
```

Transforms world coordinates to viewport / window coordinates.

Parameters

x	float
y	float

Returns

std::pair<float, float>

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/util/Camera.↩ h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/util/Camera.cpp

7.8 Model::CollisionBox Class Reference

Public Member Functions

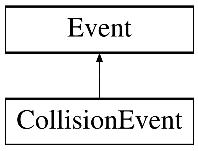
- **CollisionBox** (float left, float width, float bottom, float height)
- float **getLeft** () const
- float **getWidth** () const
- float **getBottom** () const
- float **getHeight** () const

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Entity.↩ h

7.9 CollisionEvent Class Reference

Inheritance diagram for CollisionEvent:



Public Member Functions

- **CollisionEvent** (std::shared_ptr< Model::Entity > entity, std::shared_ptr< Model::Player > player)
- void send (IEventHandler &handler) const override
- const std::shared_ptr< Model::Entity > & **getEntity** () const
- const std::shared_ptr< Model::Player > & **getPlayer** () const

Additional Inherited Members

7.9.1 Member Function Documentation

7.9.1.1 send()

```
void CollisionEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

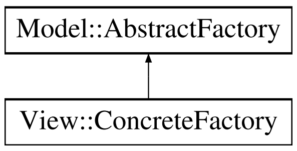
Reimplemented from Event.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.10 View::ConcreteFactory Class Reference

Inheritance diagram for View::ConcreteFactory:



Public Member Functions

- **ConcreteFactory** (const std::shared_ptr< sf::RenderWindow > &window)
- std::shared_ptr< Model::Player > createPlayer () override
- std::shared_ptr< Model::Entity > createStaticPlatform () override
- std::shared_ptr< Model::Entity > createHorizontalPlatform () override
- std::shared_ptr< Model::Entity > createVerticalPlatform () override
- std::shared_ptr< Model::Entity > createTemporaryPlatform () override
- std::shared_ptr< Model::Entity > createSpring () override
- std::shared_ptr< Model::Entity > createJetpack () override
- std::shared_ptr< Model::Entity > createBackground () override
- std::shared_ptr< Model::Score > createScore () override

7.10.1 Member Function Documentation

7.10.1.1 createBackground()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createBackground ( )  [override], [virtual]
```

Implements Model::AbstractFactory.

7.10.1.2 createHorizontalPlatform()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createHorizontalPlatform ( )  [override],
[virtual]
```

Implements Model::AbstractFactory.

7.10.1.3 createJetpack()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createJetpack ( )  [override], [virtual]
```

Implements Model::AbstractFactory.

7.10.1.4 createPlayer()

```
std::shared_ptr< Model::Player > ConcreteFactory::createPlayer ( )  [override], [virtual]
```

Implements Model::AbstractFactory.

7.10.1.5 createScore()

```
std::shared_ptr< Model::Score > ConcreteFactory::createScore ( )  [override], [virtual]
```

Implements Model::AbstractFactory.

7.10.1.6 createSpring()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createSpring ( ) [override], [virtual]
```

Implements Model::AbstractFactory.

7.10.1.7 createStaticPlatform()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createStaticPlatform ( ) [override], [virtual]
```

Implements Model::AbstractFactory.

7.10.1.8 createTemporaryPlatform()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createTemporaryPlatform ( ) [override],
[virtual]
```

Implements Model::AbstractFactory.

7.10.1.9 createVerticalPlatform()

```
std::shared_ptr< Model::Entity > ConcreteFactory::createVerticalPlatform ( ) [override],
[virtual]
```

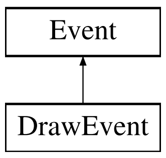
Implements Model::AbstractFactory.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/Concrete↩
Factory.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/src/Concrete↩
Factory.cpp

7.11 DrawEvent Class Reference

Inheritance diagram for DrawEvent:



Public Member Functions

- void send (IEventHandler &handler) const override

Additional Inherited Members

7.11.1 Member Function Documentation

7.11.1.1 send()

```
void DrawEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

Reimplemented from Event.

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

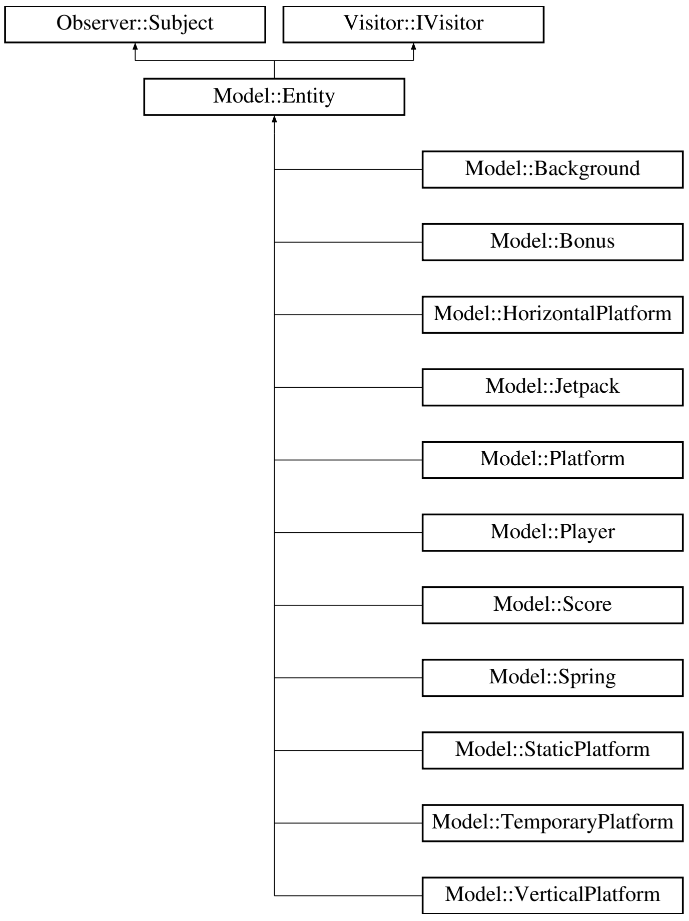
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.12 Model::Entity Class Reference

Class for Entity object.

`#include <Entity.h>`

Inheritance diagram for Model::Entity:



Public Member Functions

- **Entity** ()

  Default constructor.
- **Entity** (unsigned int score, float spawnRate)
- virtual ∼**Entity** ()=default

  Default constructor.
- float getX () const

  Get the x value of Entity object.
- float getY () const

  Get the y value of Entity object.
- void setX (float x)

  Set the x value of Entity object.
- void setY (float y)

  Set the y value of Entity object.
- float **getWidth** () const
- float getHeight () const

  Get height of Entity object.
- void setWidth (float width)

  Set width of Entity object.
- void setHeight (float height)

  Set height of Entity object.
- virtual void move (bool collision)=0

  move Entity object
- void move (float x, float y)

  Move Entity object in x and y direction.
- virtual Model::Type getType () const =0

  Get type of Entity object.
- virtual void **onDestroy** ()

  On destroy event of Entity function will be executed.
- void visit (Model::Player &player) override
- virtual void **accept** (const std::shared_ptr< Visitor::IVisitor > &visitor)
- void **setRemoveFlag** (bool flag)
- virtual bool **getRemovable** () const
- virtual bool **isRemovable** () const
- virtual bool **isBonus** () const
- void **setScore** (unsigned int score)
- virtual unsigned int **getScore** () const
- virtual float **getSpawnRate** () const
- void **setSpawnRate** (float spawnRate)

Protected Attributes

- float mX
- float mY
- float mWidth {}
- float mHeight {}
- bool **mRemoveFlag**
- unsigned int **mScore**
- float **mSpawnRate**

7.12.1 Detailed Description

Class for Entity object.

7.12.2 Member Function Documentation

7.12.2.1 getHeight()

```
float Entity::getHeight ( ) const
```

Get height of Entity object.

Returns

float

7.12.2.2 getType()

```
virtual Model::Type Model::Entity::getType ( ) const  [pure virtual]
```

Get type of Entity object.

Returns

Model::Type

Implemented in Model::Background, Model::Bonus, Model::HorizontalPlatform, Model::Jetpack, Model::Platform, Model::Player, Model::Spring, Model::StaticPlatform, Model::TemporaryPlatform, Model::VerticalPlatform, and Model::Score.

7.12.2.3 getX()

```
float Entity::getX ( ) const
```

Get the x value of Entity object.

Returns

float

7.12.2.4 getY()

```
float Entity::getY ( ) const
```

Get the y value of Entity object.

Returns

float

7.12.2.5 move() [1/2]

```
virtual void Model::Entity::move (
             bool collision )  [pure virtual]
```

move Entity object

Implemented in Model::Background, Model::HorizontalPlatform, Model::Jetpack, Model::Platform, Model::Player, Model::Spring, Model::StaticPlatform, Model::TemporaryPlatform, Model::VerticalPlatform, and Model::Score.

7.12.2.6 move() [2/2]

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

Move Entity object in x and y direction.

Parameters

x	float, added to mX
y	float, added to mY

7.12.2.7 setHeight()

```
void Entity::setHeight (
             float height )
```

Set height of Entity object.

Parameters

height	float

7.12.2.8 setWidth()

```
void Entity::setWidth (
             float width )
```

Set width of Entity object.

Parameters

width	float

7.12.2.9 setX()

```
void Entity::setX (
             float x )
```

Set the x value of Entity object.

Parameters

x	float

7.12.2.10 setY()

```
void Entity::setY (
             float y )
```

Set the y value of Entity object.

Parameters

y	float

7.12.2.11 visit()

```
void Model::Entity::visit (
             Model::Player & player )  [inline], [override], [virtual]
```

Implements Visitor::IVisitor.

7.12.3 Member Data Documentation

7.12.3.1 mHeight

```
float Model::Entity::mHeight {}  [protected]
```

Height of Entity collision box

7.12.3.2 mWidth

```
float Model::Entity::mWidth {}  [protected]
```

Width of Entity collision box

7.12.3.3 mX

```
float Model::Entity::mX  [protected]
```

X-coordinate of Entity

7.12.3.4 mY

```
float Model::Entity::mY  [protected]
```

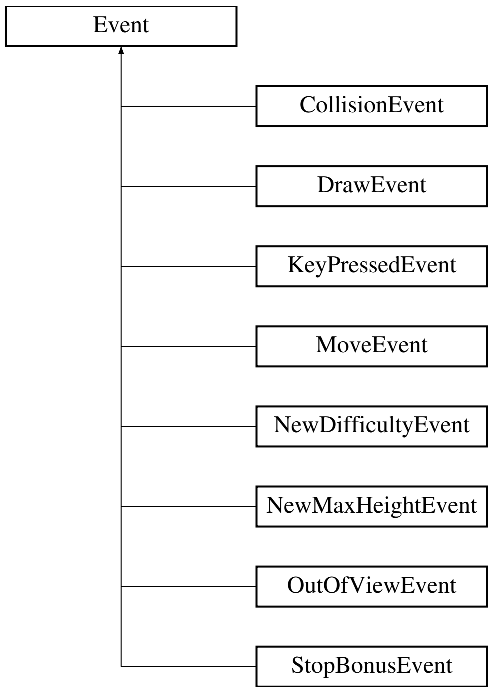
Y-coordinate of Entity

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Entity.↩
  h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Entity.cpp

7.13 Event Class Reference

Inheritance diagram for Event:



Public Member Functions

- **Event** (EventType mEvent)
- virtual void **send** (IEventHandler &handler) const
- EventType **getEvent** () const

Protected Attributes

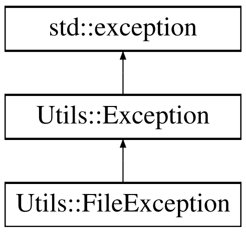
- EventType **mEvent**

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.14 Utils::Exception Class Reference

Inheritance diagram for Utils::Exception:



Public Member Functions

- **Exception** (std::string value)
- const char ∗ **what** () const noexcept override

Protected Attributes

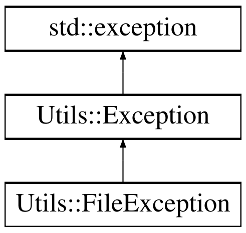
- std::string **mValue**

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/util/Exception.←
  h

7.15 Utils::FileException Class Reference

Inheritance diagram for Utils::FileException:



Public Member Functions

- **FileException** (std::string file, std::string sort)

Additional Inherited Members

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/util/Exception.←
  h

7.16 Game Class Reference

Class for Game.

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

Public Member Functions

- **Game** (unsigned int width=800, unsigned int height=1440)
- void **initializeResources** ()
- void **processEvents** ()
- void **handlePlayerInput** (sf::Keyboard::Key key, bool isPressed)
- void **render** ()
- void **run** ()
- void **drawHighScoreTable** ()

7.16.1 Detailed Description

Class for Game.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/Game.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/src/Game.cpp

7.17 HighScore Class Reference

Public Member Functions

- **HighScore** (const HighScore &)=delete
- HighScore & **operator=** (const HighScore &)=delete
- void **load** ()
- void **save** ()
- void **add** (const std::shared_ptr< HighScoreScore > &score)
- const std::vector< std::shared_ptr< HighScoreScore > > & **getScores** () const
- unsigned int **getHighScore** () const

Static Public Member Functions

- static HighScore & **getInstance** ()

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/High↩
Score.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/HighScore.cpp

7.18 HighScoreScore Struct Reference

Public Member Functions

- **HighScoreScore** (unsigned int score, std::string name)
- std::string **toString** () const

Public Attributes

- unsigned int **mScore**
- std::string **mName**

Friends

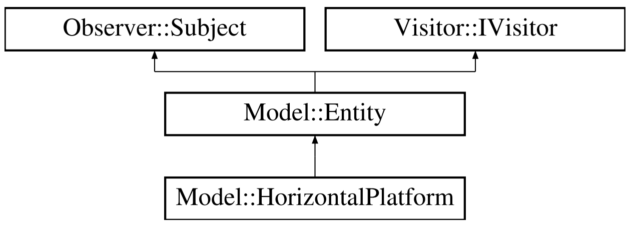
- std::ostream & **operator**<< (std::ostream &os, const HighScoreScore &score)

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/High↩
  Score.h

7.19 Model::HorizontalPlatform Class Reference

Inheritance diagram for Model::HorizontalPlatform:



Public Member Functions

- Model::Type getType () const override
  Get type of Entity object.
- void move (bool collision) override
  move Entity object
- void **initBounds** ()

Additional Inherited Members

7.19.1 Member Function Documentation

7.19.1.1 getType()

```
Model::Type HorizontalPlatform::getType ( ) const  [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.19.1.2 move()

```
void HorizontalPlatform::move (
            bool collision )  [override], [virtual]
```

move Entity object

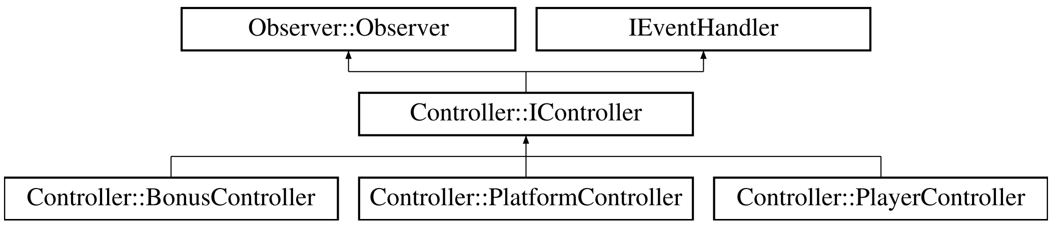
Implements Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Horizontal↩ Platform.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Horizontal↩ Platform.cpp

7.20 Controller::IController Class Reference

Inheritance diagram for Controller::IController:


Observer::Observer
IEventHandler
Controller::IController
Controller::BonusController
Controller::PlatformController
Controller::PlayerController

Public Member Functions

- **IController** (std::shared_ptr< Model::Entity > &entity)
- const std::shared_ptr< Model::Entity > & **getEntity** () const
- void onTrigger (EventType type, const std::shared_ptr< Event > &event) override

  Perform operation on trigger from Subject.
- void handleEvent (const KeyPressedEvent &event) override
- void handleEvent (const MoveEvent &event) override
- void handleEvent (const CollisionEvent &event) override

Protected Attributes

- std::shared_ptr< Model::Entity > **mEntity**

7.20.1 Member Function Documentation

7.20.1.1 handleEvent() `[1/3]`

```
void Controller::IController::handleEvent (
            const CollisionEvent & event )  [inline], [override], [virtual]
```

Reimplemented from IEventHandler.

7.20.1.2 handleEvent() `[2/3]`

```
void Controller::IController::handleEvent (
            const KeyPressedEvent & event )  [inline], [override], [virtual]
```

Reimplemented from IEventHandler.

7.20.1.3 handleEvent() `[3/3]`

```
void IController::handleEvent (
            const MoveEvent & event )  [override], [virtual]
```

Reimplemented from IEventHandler.

7.20.1.4 onTrigger()

```
void Controller::IController::onTrigger (
            EventType type,
            const std::shared_ptr< Event > & event )  [inline], [override], [virtual]
```

Perform operation on trigger from Subject.

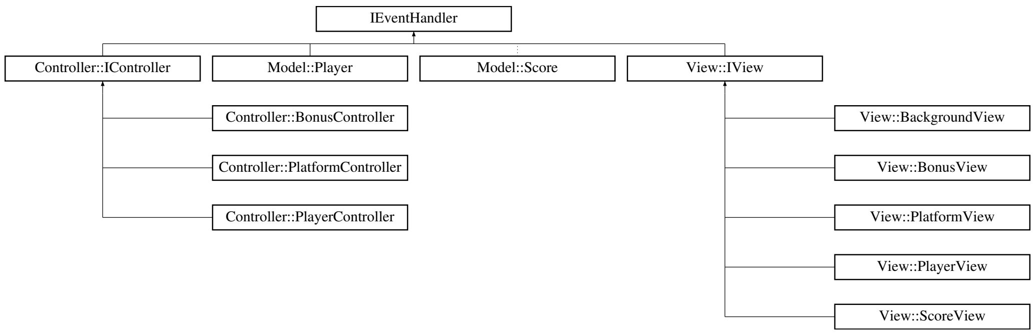
Implements Observer::Observer.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/controller/IController.↩
  h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/controller/IController.↩
  cpp

7.21 IEventHandler Class Reference

Inheritance diagram for IEventHandler:



Public Member Functions

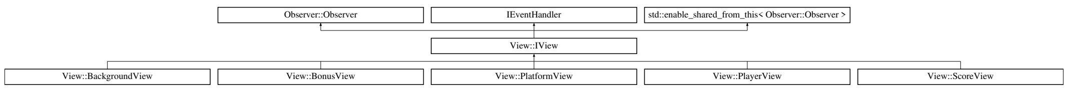
- virtual void **handleEvent** (const Event &event)
- virtual void **handleEvent** (const DrawEvent &event)
- virtual void **handleEvent** (const OutOfViewEvent &event)
- virtual void **handleEvent** (const NewMaxHeightEvent &event)
- virtual void **handleEvent** (const CollisionEvent &event)
- virtual void **handleEvent** (const KeyPressedEvent &event)
- virtual void **handleEvent** (const MoveEvent &event)
- virtual void **handleEvent** (const StopBonusEvent &event)
- virtual void **handleEvent** (const NewDifficultyEvent &event)

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.22 View::IView Class Reference

Inheritance diagram for View::IView:



Public Member Functions

- **IView** (const std::shared_ptr< Model::Entity > &entity, const std::shared_ptr< sf::RenderWindow > &window)
- void **drawCollisionBox** ()
- void onTrigger (EventType type, const std::shared_ptr< Event > &event) override

    Perform operation on trigger from Subject.
- void handleEvent (const DrawEvent &event) override
- void handleEvent (const OutOfViewEvent &event) override
- virtual void handleEvent (const CollisionEvent &event) override
- void handleEvent (const NewDifficultyEvent &event) override

Static Public Member Functions

- template<class Type >
  static void **setRainbowColor** (const std::unique_ptr< sf::Text > &object)

Protected Attributes

- std::shared_ptr< Model::Entity > **mEntity**
- std::unique_ptr< sf::Sprite > **mSprite**
- std::shared_ptr< sf::RenderWindow > **mWindow**
- std::unique_ptr< sf::Sound > **mSound**

7.22.1 Member Function Documentation

7.22.1.1 handleEvent() [1/4]

```
virtual void View::IView::handleEvent (
            const CollisionEvent & event )  [inline], [override], [virtual]
```

Reimplemented from IEventHandler.

7.22.1.2 handleEvent() [2/4]

```
void IView::handleEvent (
            const DrawEvent & event )  [override], [virtual]
```

Reimplemented from IEventHandler.

7.22.1.3 handleEvent() [3/4]

```
void View::IView::handleEvent (
            const NewDifficultyEvent & event )  [inline], [override], [virtual]
```

Reimplemented from IEventHandler.

7.22.1.4 handleEvent() [4/4]

```
void IView::handleEvent (
            const OutOfViewEvent & event )  [override], [virtual]
```

Reimplemented from IEventHandler.

7.22.1.5 onTrigger()

```
void IView::onTrigger (
            EventType type,
            const std::shared_ptr< Event > & event )  [override], [virtual]
```

Perform operation on trigger from Subject.

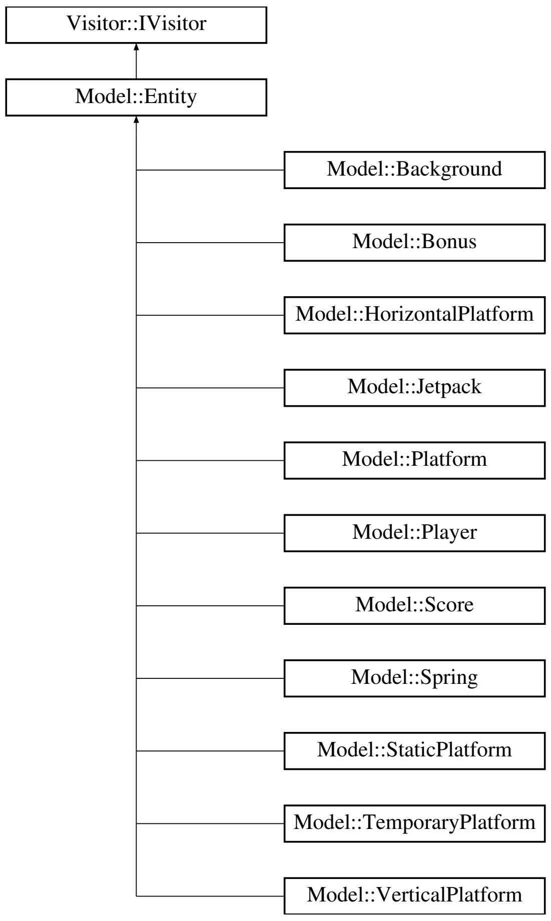
Implements Observer::Observer.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/view/IView.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/src/view/IView.cpp

7.23 Visitor::IVisitor Class Reference

Inheritance diagram for Visitor::IVisitor:



Public Member Functions

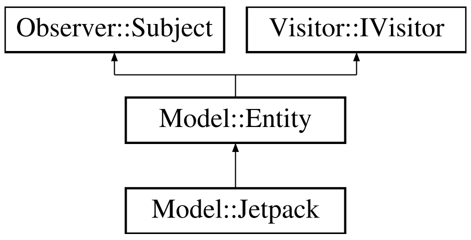
- virtual void **visit** (Model::Player &player)=0

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/IVisitor.h

7.24 Model::Jetpack Class Reference

Inheritance diagram for Model::Jetpack:



Public Member Functions

- **Jetpack** (bool started)
- Model::Type getType () const override

    Get type of Entity object.
- void move (bool collision) override

    move Entity object
- void **initBounds** ()
- void visit (Model::Player &player) override
- bool isRemovable () const override
- bool isBonus () const override

Public Attributes

- bool **mStarted**

Additional Inherited Members

7.24.1 Member Function Documentation

7.24.1.1 getType()

```
Model::Type Model::Jetpack::getType ( ) const  [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.24.1.2 isBonus()

```
bool Model::Jetpack::isBonus ( ) const  [inline], [override], [virtual]
```

Reimplemented from Model::Entity.

7.24.1.3 isRemovable()

```
bool Model::Jetpack::isRemovable ( ) const  [inline], [override], [virtual]
```

Reimplemented from Model::Entity.

7.24.1.4 move()

```
void Jetpack::move (
            bool collision )  [override], [virtual]
```

move Entity object

Implements Model::Entity.

7.24.1.5 visit()

```
void Model::Jetpack::visit (
            Model::Player & player )  [inline], [override], [virtual]
```

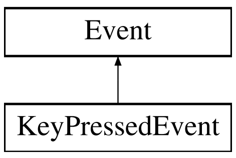
Reimplemented from Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Jetpack.↩
  h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Jetpack.↩
  cpp

7.25 KeyPressedEvent Class Reference

Inheritance diagram for KeyPressedEvent:



Public Member Functions

- **KeyPressedEvent** (std::string key, bool isPressed)
- void send (IEventHandler &handler) const override
- const std::string & **getKey** () const
- bool **isPressed** () const

Additional Inherited Members

7.25.1 Member Function Documentation

7.25.1.1 send()

```
void KeyPressedEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

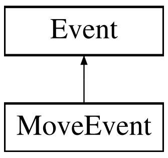
Reimplemented from Event.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.26 MoveEvent Class Reference

Inheritance diagram for MoveEvent:



Public Member Functions

- **MoveEvent** (bool collided)
- void send (IEventHandler &handler) const override
- bool **isCollided** () const

Additional Inherited Members

7.26.1 Member Function Documentation

7.26.1.1 send()

```
void MoveEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

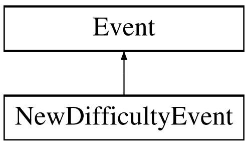
Reimplemented from Event.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.27 NewDifficultyEvent Class Reference

Inheritance diagram for NewDifficultyEvent:



Public Member Functions

- **NewDifficultyEvent** (Settings::Difficulty difficulty)
- void send (IEventHandler &handler) const override
- Settings::Difficulty **getDifficulty** () const

Additional Inherited Members

7.27.1 Member Function Documentation

7.27.1.1 send()

```
void NewDifficultyEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

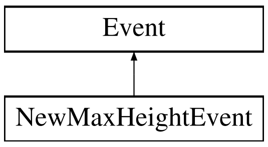
Reimplemented from Event.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.28 NewMaxHeightEvent Class Reference

Inheritance diagram for NewMaxHeightEvent:



Public Member Functions

- **NewMaxHeightEvent** (float lastHeight, float newHeight)
- void send (IEventHandler &handler) const override
- float **getLastHeight** () const
- float **getNewHeight** () const

Additional Inherited Members

7.28.1 Member Function Documentation

7.28.1.1 send()

```
void NewMaxHeightEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

Reimplemented from Event.

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

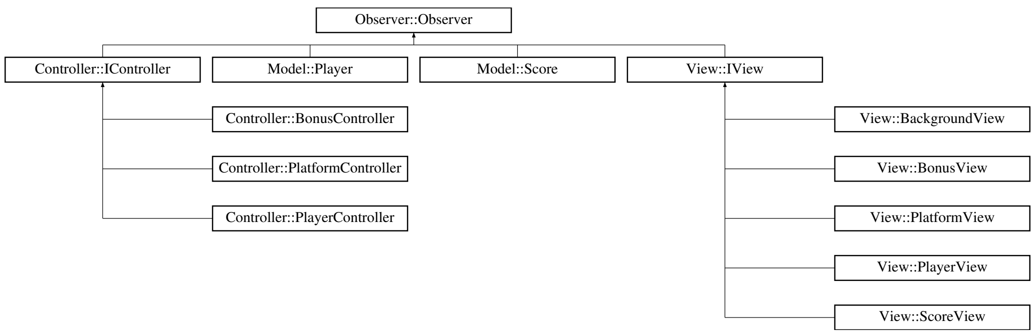
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.29 Observer::Observer Class Reference

Class for Observer of Observer pattern.

`#include <Observer.h>`

Inheritance diagram for Observer::Observer:



Public Member Functions

- **Observer** ()=default

  Default constructor.
- virtual ∼**Observer** ()=default

  Default destructor.
- virtual void onTrigger (EventType type, const std::shared_ptr< Event > &event)=0

  Perform operation on trigger from Subject.

7.29.1 Detailed Description

Class for Observer of Observer pattern.

7.29.2 Member Function Documentation

7.29.2.1 onTrigger()

```
virtual void Observer::Observer::onTrigger (
             EventType type,
             const std::shared_ptr< Event > & event )  [pure virtual]
```

Perform operation on trigger from Subject.

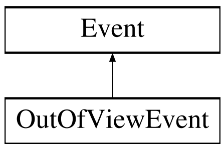
Implemented in Controller::IController, Model::Player, Model::Score, and View::IView.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Observer.h

7.30 OutOfViewEvent Class Reference

Inheritance diagram for OutOfViewEvent:



Public Member Functions

- void send (IEventHandler &handler) const override

Additional Inherited Members

7.30.1 Member Function Documentation

7.30.1.1 send()

```
void OutOfViewEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

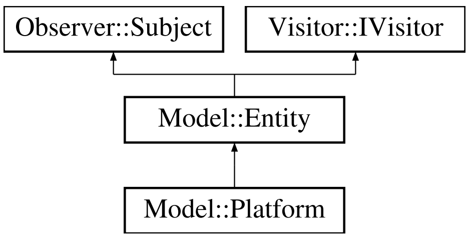
Reimplemented from Event.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.31 Model::Platform Class Reference

Inheritance diagram for Model::Platform:



Public Member Functions

- **Platform** (Model::Type sort)
- Model::Type getType () const override

  Get type of Entity object.
- void move (bool collision) override

  move Entity object
- void **initBounds** ()

Additional Inherited Members

7.31.1 Member Function Documentation

7.31.1.1 getType()

```
Model::Type Model::Platform::getType ( ) const  [inline], [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.31.1.2 move()

```
void Platform::move (
             bool collision )  [override], [virtual]
```

move Entity object

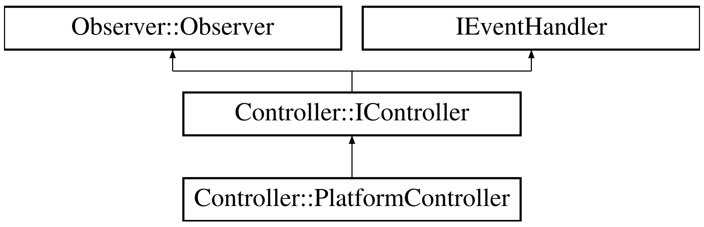
Implements Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Platform.↩ h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Platform.↩ cpp

7.32 Controller::PlatformController Class Reference

Inheritance diagram for Controller::PlatformController:



Public Member Functions

- **PlatformController** (std::shared_ptr< Model::Entity > &entity)

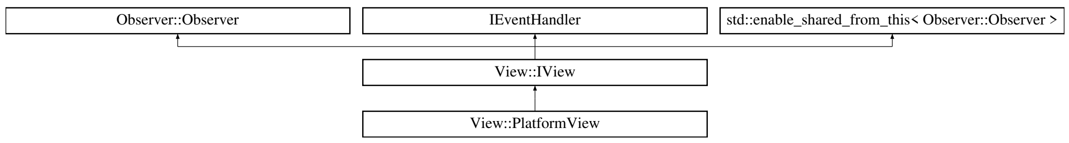
Additional Inherited Members

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/controller/Platform↩ Controller.h

7.33 View::PlatformView Class Reference

Inheritance diagram for View::PlatformView:



Public Member Functions

- **PlatformView** (const std::shared_ptr< Model::Entity > &entity, const std::shared_ptr< sf::RenderWindow > &window)
- void handleEvent (const CollisionEvent &event) override

Additional Inherited Members

7.33.1 Member Function Documentation

7.33.1.1 handleEvent()

```
void View::PlatformView::handleEvent (
            const CollisionEvent & event )  [inline], [override], [virtual]
```

Reimplemented from View::IView.

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

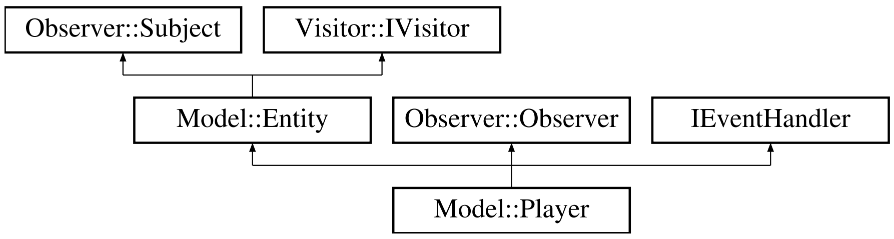
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/view/Platform↩ View.h

7.34 Model::Player Class Reference

Class for Player object, derives from Entity.

```
#include <Player.h>
```

Inheritance diagram for Model::Player:



Public Member Functions

- Model::Type getType () const override

    Get type of Entity object.
- void move (bool collision) override

    move Entity object
- const std::pair< float, float > & **getVelocity** () const
- void **setVelocity** (const std::pair< float, float > &velocity)
- const std::pair< float, float > & **getDirection** () const
- void **setDirection** (const std::pair< float, float > &direction)
- const float **getMaxVelocity** () const
- const float **getMaxAcceleration** () const
- const float **getDrag** () const
- void **setDrag** (float drag)
- bool **isMovingUp** () const
- void **setIsMovingUp** (bool isMovingUp)
- bool **isMovingDown** () const
- void **setIsMovingDown** (bool isMovingDown)
- bool **isMovingLeft** () const
- void **setIsMovingLeft** (bool isMovingLeft)
- bool **isMovingRight** () const
- void **setIsMovingRight** (bool isMovingRight)
- void accept (const std::shared_ptr< Visitor::IVisitor > &visitor) override
- void onTrigger (EventType type, const std::shared_ptr< Event > &event) override

    Perform operation on trigger from Subject.
- void handleEvent (const StopBonusEvent &event) override

Additional Inherited Members

7.34.1 Detailed Description

Class for Player object, derives from Entity.

7.34.2 Member Function Documentation

7.34.2.1 accept()

```
void Model::Player::accept (
             const std::shared_ptr< Visitor::IVisitor > & visitor )  [inline], [override],
[virtual]
```

Reimplemented from Model::Entity.

7.34.2.2 getType()

```
Model::Type Model::Player::getType ( ) const  [inline], [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.34.2.3 handleEvent()

```
void Model::Player::handleEvent (
             const StopBonusEvent & event )  [inline], [override], [virtual]
```

Reimplemented from IEventHandler.

7.34.2.4 move()

```
void Player::move (
             bool collision )  [override], [virtual]
```

move Entity object

Implements Model::Entity.

7.34.2.5 onTrigger()

```
void Model::Player::onTrigger (
             EventType type,
             const std::shared_ptr< Event > & event )  [inline], [override], [virtual]
```

Perform operation on trigger from Subject.

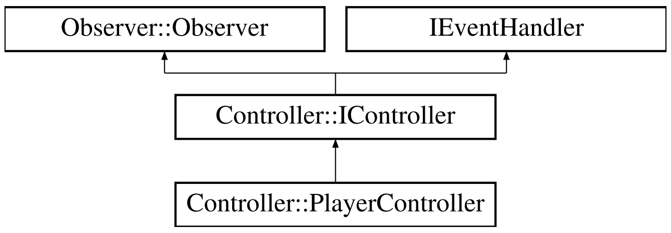
Implements Observer::Observer.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Player.↩
h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Player.cpp

7.35 Controller::PlayerController Class Reference

Inheritance diagram for Controller::PlayerController:



Public Member Functions

- **PlayerController** (std::shared_ptr< Model::Entity > &entity)
- **PlayerController** (std::shared_ptr< Model::Player > &entity)
- void handleEvent (const KeyPressedEvent &event) override
- void handleEvent (const MoveEvent &event) override
- void handleEvent (const CollisionEvent &event) override

Additional Inherited Members

7.35.1 Member Function Documentation

7.35.1.1 handleEvent() [1/3]

```
void PlayerController::handleEvent (
             const CollisionEvent & event )  [override], [virtual]
```

Reimplemented from Controller::IController.

7.35.1.2 handleEvent() [2/3]

```
void PlayerController::handleEvent (
            const KeyPressedEvent & event )  [override], [virtual]
```

Reimplemented from Controller::IController.

7.35.1.3 handleEvent() [3/3]

```
void PlayerController::handleEvent (
            const MoveEvent & event )  [override], [virtual]
```

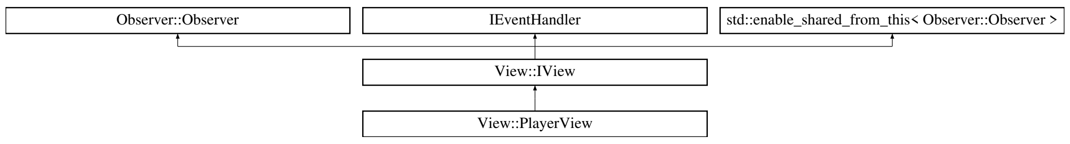
Reimplemented from Controller::IController.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/controller/Player↩ Controller.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/controller/Player↩ Controller.cpp

7.36 View::PlayerView Class Reference

Inheritance diagram for View::PlayerView:



Public Member Functions

- **PlayerView** (const std::shared_ptr< Model::Entity > &entity, const std::shared_ptr< sf::RenderWindow > &window)

Additional Inherited Members

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/view/Player↩ View.h

7.37 Utils::Random Class Reference

Class for Random.

```
#include <Random.h>
```

Public Member Functions

- ∼**Random** ()=default

  Default destructor.
- **Random** (const Random &)=delete

  Deleted copy constructor.
- Random & operator= (const Random &)=delete

  Deleted assignment operator.
- float random (float a, float b)

  Get random float in given interval.

Static Public Member Functions

- static Random & getInstance ()

  Get instance of Random.

7.37.1 Detailed Description

Class for Random.

7.37.2 Member Function Documentation

7.37.2.1 getInstance()

```
Random & Random::getInstance ( )  [static]
```

Get instance of Random.

Returns

Random

7.37.2.2 operator=()

```
Random & Utils::Random::operator= (
            const Random & )  [delete]
```

Deleted assignment operator.

Returns

Random

7.37.2.3 random()

```
float Random::random (
            float a,
            float b )
```

Get random float in given interval.

Parameters

a	float - begin interval
b	float - end interval

Returns

float

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/util/Random.↩
h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/util/Random.cpp

7.38 Utils::Resourceholder< Type > Struct Template Reference

Public Member Functions

- **Resourceholder** (std::string path)
- void **insert** (Model::Type type, const std::string &subPath)
- std::shared_ptr< Type > & **get** (Model::Type type)

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/util/Resourcemanager.↩
h

7.39 Utils::Resourcemanager Class Reference

Public Member Functions

- **Resourcemanager** (const Resourcemanager &)=delete
- Resourcemanager & **operator=** (const Resourcemanager &)=delete
- void **addTexture** (Model::Type type, const std::string &subPath)
- void **addFont** (Model::Type type, const std::string &subPath)
- void **addSound** (Model::Type type, const std::string &subPath)
- const std::shared_ptr< Resourceholder< sf::Texture > > & **getTextures** () const
- const std::shared_ptr< Resourceholder< sf::Font > > & **getFonts** () const
- const std::shared_ptr< Resourceholder< sf::SoundBuffer > > & **getSounds** () const

Static Public Member Functions

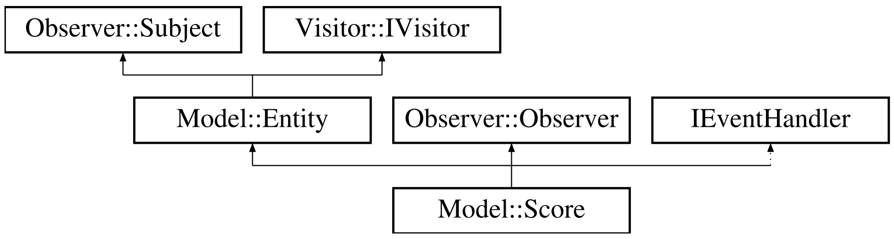
- static Resourcemanager & **getInstance** ()

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/util/Resourcemanager.↩
h

7.40 Model::Score Class Reference

Inheritance diagram for Model::Score:



Public Member Functions

- Model::Type getType () const override

  Get type of Entity object.
- void **setScore** (unsigned int score)
- void onTrigger (EventType type, const std::shared_ptr< Event > &event) override

  Perform operation on trigger from Subject.
- void handleEvent (const NewMaxHeightEvent &event) override
- void handleEvent (const CollisionEvent &event) override
- void move (bool collision) override

  move Entity object
- unsigned int getScore () const override

Additional Inherited Members

7.40.1 Member Function Documentation

7.40.1.1 getScore()

```
unsigned int Model::Score::getScore ( ) const  [inline], [override], [virtual]
```

Reimplemented from Model::Entity.

7.40.1.2 getType()

```
Model::Type Model::Score::getType ( ) const  [inline], [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.40.1.3 handleEvent() [1/2]

```
void Score::handleEvent (
             const CollisionEvent & event )  [override], [virtual]
```

Reimplemented from IEventHandler.

7.40.1.4 handleEvent() [2/2]

```
void Score::handleEvent (
             const NewMaxHeightEvent & event )  [override], [virtual]
```

Reimplemented from IEventHandler.

7.40.1.5 move()

```
void Model::Score::move (
            bool collision )  [inline], [override], [virtual]
```

move Entity object

Implements Model::Entity.

7.40.1.6 onTrigger()

```
void Model::Score::onTrigger (
            EventType type,
            const std::shared_ptr< Event > & event )  [inline], [override], [virtual]
```

Perform operation on trigger from Subject.

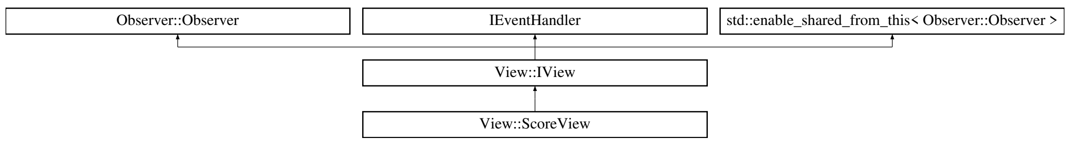
Implements Observer::Observer.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Score.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/Score.cpp

7.41 View::ScoreView Class Reference

Inheritance diagram for View::ScoreView:



Public Member Functions

- **ScoreView** (std::shared_ptr< Model::Score > &entity, const std::shared_ptr< sf::RenderWindow > &window)

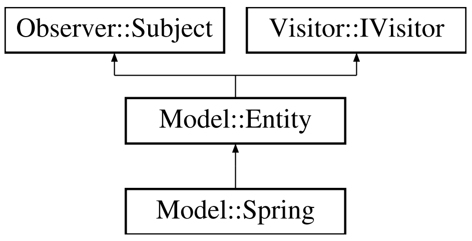
Additional Inherited Members

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/include/view/Score↩ View.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/game/src/view/Score↩ View.cpp

7.42 Model::Spring Class Reference

Inheritance diagram for Model::Spring:



Public Member Functions

- Model::Type getType () const override

    Get type of Entity object.
- void move (bool collision) override

    move Entity object
- void **initBounds** ()
- void visit (Model::Player &player) override
- bool isBonus () const override
- bool isRemovable () const override

Additional Inherited Members

7.42.1 Member Function Documentation

7.42.1.1 getType()

```
Model::Type Model::Spring::getType ( ) const  [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.42.1.2 isBonus()

```
bool Model::Spring::isBonus ( ) const  [inline], [override], [virtual]
```

Reimplemented from Model::Entity.

7.42.1.3 isRemovable()

```
bool Model::Spring::isRemovable ( ) const  [inline], [override], [virtual]
```

Reimplemented from Model::Entity.

7.42.1.4 move()

```
void Spring::move (
            bool collision )  [override], [virtual]
```

move Entity object

Implements Model::Entity.

7.42.1.5 visit()

```
void Model::Spring::visit (
            Model::Player & player )  [inline], [override], [virtual]
```

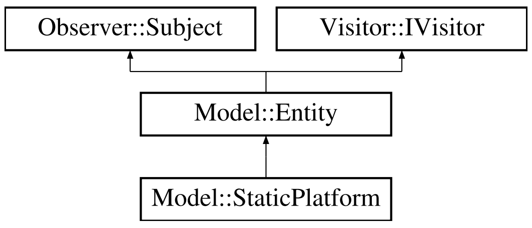
Reimplemented from Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Spring.↩ h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Spring.↩ cpp

7.43 Model::StaticPlatform Class Reference

Inheritance diagram for Model::StaticPlatform:



Public Member Functions

- Model::Type getType () const override

  Get type of Entity object.
- void move (bool collision) override

  move Entity object

Additional Inherited Members

7.43.1 Member Function Documentation

7.43.1.1 getType()

```
Model::Type StaticPlatform::getType ( ) const  [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.43.1.2 move()

```
void StaticPlatform::move (
            bool collision )  [override], [virtual]
```

move Entity object

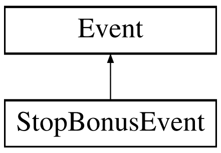
Implements Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Static↩ Platform.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Static↩ Platform.cpp

7.44 StopBonusEvent Class Reference

Inheritance diagram for StopBonusEvent:



Public Member Functions

- **StopBonusEvent** (std::shared_ptr< Model::Entity > bonus)
- void send (IEventHandler &handler) const override
- const std::shared_ptr< Model::Entity > & **getBonus** () const

Additional Inherited Members

7.44.1 Member Function Documentation

7.44.1.1 send()

```
void StopBonusEvent::send (
            IEventHandler & handler ) const  [inline], [override], [virtual]
```

Reimplemented from Event.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Event.h

7.45 Utils::Stopwatch Class Reference

Class for Stopwatch.

```
#include <Stopwatch.h>
```

Public Member Functions

- **Stopwatch** ()=default

  Private default constructor.
- ∼**Stopwatch** ()=default

  Default destructor.
- **Stopwatch** (const Stopwatch &)=delete

  Deleted copy constructor.
- Stopwatch & operator= (const Stopwatch &)=delete

  Deleted assignment operator.
- void **start** ()

  Start Stopwatch.
- float lap ()

  Lap one round and return found delta.
- float getDelta () const

  Get latest delta.
- void **addTimer** (unsigned int key, float amount)
- bool **checkTimer** (unsigned int key)

Static Public Member Functions

- static Stopwatch & getInstance ()
  Get instance of Stopwatch.

Public Attributes

- std::shared_ptr< Model::Entity > **mPlayer**

7.45.1 Detailed Description

Class for Stopwatch.

7.45.2 Member Function Documentation

7.45.2.1 getDelta()

```
float Stopwatch::getDelta ( ) const
```

Get latest delta.

Returns

float

7.45.2.2 getInstance()

```
Stopwatch & Stopwatch::getInstance ( ) [static]
```

Get instance of Stopwatch.

Returns

Stopwatch

7.45.2.3 lap()

```
float Stopwatch::lap ( )
```

Lap one round and return found delta.

Returns

float

7.45.2.4 operator=()

```
Stopwatch & Utils::Stopwatch::operator= (
             const Stopwatch &  )  [delete]
```

Deleted assignment operator.

Returns

Stopwatch

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

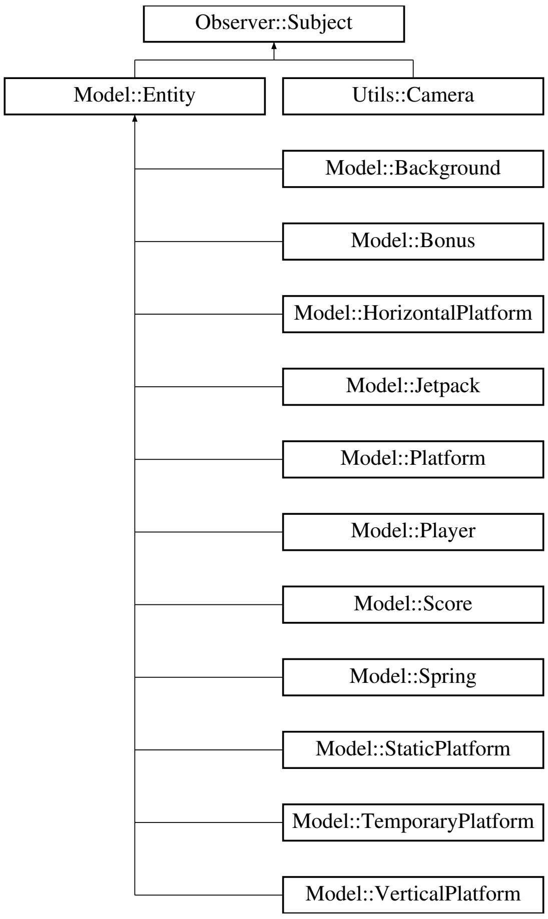
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/util/Stopwatch.↩ h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/util/Stopwatch.↩ cpp

7.46 Observer::Subject Class Reference

Class for Subject / Observable of observer pattern.

```
#include <Subject.h>
```

Inheritance diagram for Observer::Subject:



Public Member Functions

- **Subject** ()=default

  Default constructor.
- virtual ∼**Subject** ()=default

  Default destructor.
- void add (const std::shared_ptr< Observer > &observer)

  Register Observer to mObservers.
- const std::vector< std::shared_ptr< Observer > > & **getObservers** () const
- void **clear** ()

  Clear all Observers from mObservers.
- void **trigger** (EventType type, const std::shared_ptr< Event > &event) const

  Trigger registered Observers.

7.46.1 Detailed Description

Class for Subject / Observable of observer pattern.

7.46.2 Member Function Documentation

7.46.2.1 add()

```
void Observer::Subject::add (
            const std::shared_ptr< Observer > & observer )  [inline]
```

Register Observer to mObservers.

Parameters

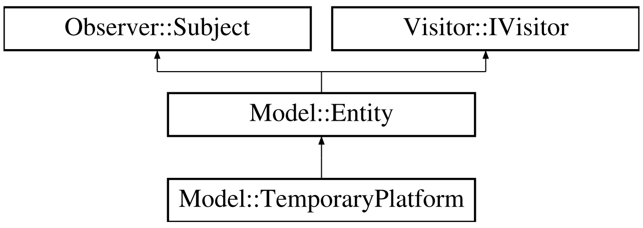
observer	Observer to be added

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/Subject.h

7.47 Model::TemporaryPlatform Class Reference

Inheritance diagram for Model::TemporaryPlatform:



Public Member Functions

- Model::Type getType () const override

  Get type of Entity object.
- void move (bool collision) override

  move Entity object

Additional Inherited Members

7.47.1 Member Function Documentation

7.47.1.1 getType()

```
Model::Type TemporaryPlatform::getType ( ) const  [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.47.1.2 move()

```
void TemporaryPlatform::move (
            bool collision )  [override], [virtual]
```

move Entity object

Implements Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Temporary↩ Platform.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Temporary↩ Platform.cpp

7.48 Utils::Utilities Class Reference

Class for Utilities.

```
#include <Utilities.h>
```

Static Public Member Functions

- static bool checkCollision (const std::shared_ptr< Model::Entity > &l, const std::shared_ptr< Model::Entity > &r)

    Check if there is a collision between two Entities.
- static bool checkWeight (float &rand, float weight)

    Check random spawn rate of total chance.

7.48.1 Detailed Description

Class for Utilities.

7.48.2 Member Function Documentation

7.48.2.1 checkCollision()

```
bool Utilities::checkCollision (
            const std::shared_ptr< Model::Entity > & l,
            const std::shared_ptr< Model::Entity > & r )  [static]
```

Check if there is a collision between two Entities.

Parameters

l	first entity
r	second entity

Returns

bool

7.48.2.2 checkWeight()

```
bool Utilities::checkWeight (
            float & rand,
            float weight )  [static]
```

Check random spawn rate of total chance.

Parameters

rand	float - random spawn rate
weight	float - total chance

Returns

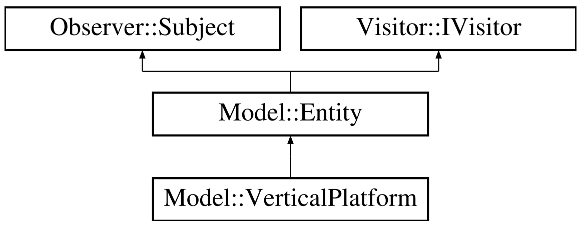
true if rand <= weight

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/util/Utilities.↩ h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/util/Utilities.cpp

7.49 Model::VerticalPlatform Class Reference

Inheritance diagram for Model::VerticalPlatform:



Public Member Functions

- Model::Type getType () const override

    Get type of Entity object.
- void move (bool collision) override

    move Entity object
- void **initBounds** ()

Additional Inherited Members

7.49.1 Member Function Documentation

7.49.1.1 getType()

```
Model::Type VerticalPlatform::getType ( ) const  [override], [virtual]
```

Get type of Entity object.

Returns

Model::Type

Implements Model::Entity.

7.49.1.2 move()

```
void VerticalPlatform::move (
            bool collision )  [override], [virtual]
```

move Entity object

Implements Model::Entity.

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/model/Vertical↵
Platform.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/model/Vertical↵
Platform.cpp

7.50 World Class Reference

Class for World, holds all the entities and is used for the overall game logic.

```
#include <World.h>
```

Public Member Functions

- **World** (std::shared_ptr< Model::AbstractFactory > &factory)
- void **initWorld** ()

  Initialize starting-world.
- void events (const std::string &move, bool isPressed) const

  Process events such as player input.
- void **update** ()

  Gets called every tick from Game loop, updates ever Model.
- void **render** () const

  Render all entities on screen.
- void **generateEntity** ()

  Generate new Entity.
- void spawnPlatform (float x, float y)

  Spawn random Platform object into World.
- void spawnBonus (float x, float y)

  Spawn random Bonus object AND Platform object into World.
- void spawnEntity (float x, float y, Model::Type type)

  Spawn provided Entity into World.
- bool **checkDifficulty** ()
- void addEntity (const std::shared_ptr< Model::Entity > &entity)

  Add entity.
- void **removeEntities** ()

  Remove unused Entities or those that are out of view.
- void **destroy** ()
- const std::shared_ptr< Model::Score > & **getScore** () const
- bool **isPlaying** () const

7.50.1 Detailed Description

Class for World, holds all the entities and is used for the overall game logic.

7.50.2 Member Function Documentation

7.50.2.1 addEntity()

```
void World::addEntity (
             const std::shared_ptr< Model::Entity > & entity )
```

Add entity.

Parameters

entity	Entity to be added to mEntities

7.50.2.2 events()

```
void World::events (
             const std::string & move,
             bool isPressed ) const
```

Process events such as player input.

Parameters

move	std::string - which key is pressed
isPressed	bool - is key pressed

7.50.2.3 spawnBonus()

```
void World::spawnBonus (
             float x,
             float y )
```

Spawn random Bonus object AND Platform object into World.

Parameters

x	float
y	float

7.50.2.4 spawnEntity()

```
void World::spawnEntity (
             float x,
             float y,
             Model::Type type )
```

Spawn provided Entity into World.

Parameters

x	float
y	float
type	Model::Type - Sort of Entity to be spawned

7.50.2.5 spawnPlatform()

```
void World::spawnPlatform (
             float x,
             float y )
```

Spawn random Platform object into World.

Parameters

x	float
y	float

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

- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/include/World.h
- /Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/engine/src/World.cpp

Chapter 8

File Documentation

8.1 AbstractFactory.h

```cpp
//
// Created by Pablo Deputter on 21/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_ABSTRACTFACTORY_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_ABSTRACTFACTORY_H

#include "Score.h"
#include "controller/IController.h"
#include "model/Entity.h"

namespace Model {
// Used by world to create new entities without knowing anything sfml-related (VIEW)

// The game class provides pointer to concrete factory to world, so it can create
// entities that have correct view attached
class AbstractFactory
{
public:
        AbstractFactory() = default;

        virtual ~AbstractFactory() = default;

        virtual std::shared_ptr<Model::Player> createPlayer() = 0;

        virtual std::shared_ptr<Model::Entity> createStaticPlatform() = 0;

        virtual std::shared_ptr<Model::Entity> createHorizontalPlatform() = 0;

        virtual std::shared_ptr<Model::Entity> createVerticalPlatform() = 0;

        virtual std::shared_ptr<Model::Entity> createTemporaryPlatform() = 0;

        virtual std::shared_ptr<Model::Entity> createSpring() = 0;

        virtual std::shared_ptr<Model::Entity> createJetpack() = 0;

        virtual std::shared_ptr<Model::Entity> createBackground() = 0;

        virtual std::shared_ptr<Model::Score> createScore() = 0;
};
} // namespace Model

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_ABSTRACTFACTORY_H
```

8.2 BonusController.h

```cpp
//
// Created by Pablo Deputter on 07/12/2021.
//

#ifndef DOODLEJUMP_BONUSCONTROLLER_H
#define DOODLEJUMP_BONUSCONTROLLER_H

```

```
#include "IController.h"

namespace Controller {

class BonusController : public IController
{
public:
        BonusController(std::shared_ptr<Model::Entity>& entity) : IController(entity) {}

        BonusController() = default;

        ~BonusController() override = default;
};
} // namespace Controller

#endif // DOODLEJUMP_BONUSCONTROLLER_H
```

8.3 IController.h

```
//
// Created by Pablo Deputter on 19/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_ICONTROLLER_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_ICONTROLLER_H

#include "model/Entity.h"
#include "model/Player.h"

#include <memory>

namespace Controller {

class IController : public Observer::Observer, public IEventHandler
{
protected:
        std::shared_ptr<Model::Entity> mEntity;

public:
        IController() = default;

        IController(std::shared_ptr<Model::Entity>& entity) : mEntity(entity) {}

        virtual ~IController() override = default;

        const std::shared_ptr<Model::Entity>& getEntity() const { return mEntity; }

        void onTrigger(EventType type, const std::shared_ptr<Event>& event) override {
    event->send(*this); }

        void handleEvent(const KeyPressedEvent& event) override {}

        void handleEvent(const MoveEvent& event) override;

        void handleEvent(const CollisionEvent& event) override {}
};
} // namespace Controller

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_ICONTROLLER_H
```

8.4 PlatformController.h

```
//
// Created by Pablo Deputter on 20/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORMCONTROLLER_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORMCONTROLLER_H

#include "IController.h"

namespace Controller {

class PlatformController : public IController
{
public:
        PlatformController(std::shared_ptr<Model::Entity>& entity) : IController(entity) {}

        PlatformController() = default;

        ~PlatformController() override = default;
};
} // namespace Controller

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORMCONTROLLER_H
```

8.5 PlayerController.h

```
//
// Created by Pablo Deputter on 19/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYERCONTROLLER_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYERCONTROLLER_H

#include "IController.h"

namespace Controller {

class PlayerController : public IController
{
public:
        explicit PlayerController(std::shared_ptr<Model::Entity>& entity) : IController(entity) {}

        explicit PlayerController(std::shared_ptr<Model::Player>& entity) { mEntity = entity; }

        PlayerController() = default;

        ~PlayerController() override = default;

        void handleEvent(const KeyPressedEvent& event) override;

        void handleEvent(const MoveEvent& event) override;

        void handleEvent(const CollisionEvent& event) override;
};
} // namespace Controller

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYERCONTROLLER_H
```

8.6 Event.h

```
//
// Created by Pablo Deputter on 08/12/2021.
//

#ifndef DOODLEJUMP_EVENT_H
#define DOODLEJUMP_EVENT_H

#include "Settings.h"

#include <iostream>
#include <memory>
#include <string>
#include <utility>

namespace Model {
class Entity;
class Player;
} // namespace Model

enum class EventType
{
        OUT_OF_VIEW = 0,
        DRAW,
        NEW_MAX_HEIGHT,
        COLLISION,
        KEY_PRESSED,
        MOVE,
        STOP_BONUS,
        NEW_DIFFICULTY
};

class Event;
class DrawEvent;
class OutOfViewEvent;
class NewMaxHeightEvent;
class CollisionEvent;
class KeyPressedEvent;
class MoveEvent;
class StopBonusEvent;
class NewDifficultyEvent;

class IEventHandler
{
public:
        virtual void handleEvent(const Event& event) {}

        virtual void handleEvent(const DrawEvent& event) {}

        virtual void handleEvent(const OutOfViewEvent& event) {}

        virtual void handleEvent(const NewMaxHeightEvent& event) {}

        virtual void handleEvent(const CollisionEvent& event) {}

        virtual void handleEvent(const KeyPressedEvent& event) {}

        virtual void handleEvent(const MoveEvent& event) {}

        virtual void handleEvent(const StopBonusEvent& event) {}

        virtual void handleEvent(const NewDifficultyEvent& event) {}
};

class Event
{
protected:
        EventType mEvent;

public:
        explicit Event(EventType mEvent) : mEvent(mEvent) {}

        Event() = default;

        virtual ~Event() = default;

        virtual void send(IEventHandler& handler) const { handler.handleEvent(*this); }

        [[nodiscard]] EventType getEvent() const { return mEvent; }
};

class DrawEvent : public Event
{
public:
        DrawEvent() : Event() {}

        ~DrawEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }
};

class OutOfViewEvent : public Event
{
public:
        OutOfViewEvent() : Event() {}

        ~OutOfViewEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }
};

class NewMaxHeightEvent : public Event
{
private:
        float mLastHeight;
        float mNewHeight;

public:
        NewMaxHeightEvent(float lastHeight, float newHeight) : Event(), mLastHeight(lastHeight),
    mNewHeight(newHeight)
        {
        }

        ~NewMaxHeightEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }

        float getLastHeight() const { return mLastHeight; }

        float getNewHeight() const { return mNewHeight; }
};

class CollisionEvent : public Event
{
```

```
private:
        std::shared_ptr<Model::Entity> mEntity;
        std::shared_ptr<Model::Player> mPlayer;

public:
        explicit CollisionEvent(std::shared_ptr<Model::Entity> entity, std::shared_ptr<Model::Player>
    player)
                : Event(), mEntity(std::move(entity)), mPlayer(std::move(player))
        {
        }

        ~CollisionEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }

        const std::shared_ptr<Model::Entity>& getEntity() const { return mEntity; }

        const std::shared_ptr<Model::Player>& getPlayer() const { return mPlayer; }
};

class KeyPressedEvent : public Event
{
private:
        std::string mKey;
        bool mIsPressed;

public:
        KeyPressedEvent(std::string key, bool isPressed) : Event(), mKey(std::move(key)),
    mIsPressed(isPressed) {}

        ~KeyPressedEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }

        const std::string& getKey() const { return mKey; }

        bool isPressed() const { return mIsPressed; }
};

class MoveEvent : public Event
{
private:
        bool mCollided;

public:
        explicit MoveEvent(bool collided) : Event(), mCollided(collided) {}

        ~MoveEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }

        bool isCollided() const { return mCollided; }
};

class StopBonusEvent : public Event
{
private:
        std::shared_ptr<Model::Entity> mBonus;

public:
        explicit StopBonusEvent(std::shared_ptr<Model::Entity> bonus) : Event(),
    mBonus(std::move(bonus)) {}

        ~StopBonusEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }

        const std::shared_ptr<Model::Entity>& getBonus() const { return mBonus; }
};

class NewDifficultyEvent : public Event
{
private:
        Settings::Difficulty mDifficulty;

public:
        explicit NewDifficultyEvent(Settings::Difficulty difficulty) : Event(), mDifficulty(difficulty)
    {}

        ~NewDifficultyEvent() override = default;

        void send(IEventHandler& handler) const override { handler.handleEvent(*this); }

        [[nodiscard]] Settings::Difficulty getDifficulty() const { return mDifficulty; }
};

#endif // DOODLEJUMP_EVENT_H
```

8.7 HighScore.h

```
//
// Created by Pablo Deputter on 13/12/2021.
//

#ifndef DOODLEJUMP_HIGHSCORE_H
#define DOODLEJUMP_HIGHSCORE_H

#include "util/Exception.h"

#include <fstream>
#include <iostream>
#include <memory>
#include <ostream>
#include <string>
#include <vector>

struct HighScoreScore
{
        unsigned int mScore;
        std::string mName;

        HighScoreScore() : mScore(0), mName(std::string()) {}

        HighScoreScore(unsigned int score, std::string name) : mScore(score), mName(std::move(name)) {}

        virtual ~HighScoreScore() = default;

        [[nodiscard]] std::string toString() const { return mName + " - " + std::to_string(mScore); }

        friend std::ostream& operator«(std::ostream& os, const HighScoreScore& score)
        {
                os « score.mName « " - " « score.mScore « "\n";
                return os;
        }
};

class HighScore
{
private:
        std::string mPath;
        unsigned int mQuantity;
        std::vector<std::shared_ptr<HighScoreScore» mScores;

        HighScore() = default;

        HighScore(std::string path, unsigned int quantity) : mPath(path), mQuantity(quantity)
        {
                try {
                        load();
                } catch (const std::exception& exc) {
                        std::cerr « exc.what();
                }
        }

public:
        HighScore(const HighScore&) = delete;

        HighScore& operator=(const HighScore&) = delete;

        ~HighScore() { save(); }

        static HighScore& getInstance();

        void load();

        void save();

        void add(const std::shared_ptr<HighScoreScore>& score);

        [[nodiscard]] const std::vector<std::shared_ptr<HighScoreScore»& getScores() const { return
     mScores; }

        [[nodiscard]] unsigned int getHighScore() const
        {
                if (mScores.empty()) {
                        return 0;
                }
                return mScores.front()->mScore;
        }
};

#endif // DOODLEJUMP_HIGHSCORE_H
```

8.8 IVisitor.h

```
//
// Created by Pablo Deputter on 07/12/2021.
//

#ifndef DOODLEJUMP_IVISITOR_H
#define DOODLEJUMP_IVISITOR_H

#include <memory>

namespace Model {
class Player;
}

namespace Visitor {

class IVisitor
{
public:
        virtual void visit(Model::Player& player) = 0;
};
} // namespace Visitor

#endif // DOODLEJUMP_IVISITOR_H
```

8.9 Background.h

```
//
// Created by Pablo Deputter on 29/11/2021.
//

#ifndef DOODLEJUMP_BACKGROUND_H
#define DOODLEJUMP_BACKGROUND_H

#include "Entity.h"

namespace Model {

class Background : public Entity
{
private:
public:
        Background() = default;

        ~Background() override = default;

        Model::Type getType() const override { return Model::Type::eBackground; }

        void move(bool collision) override {}
};
} // namespace Model

#endif // DOODLEJUMP_BACKGROUND_H
```

8.10 Bonus.h

```
//
// Created by Pablo Deputter on 16/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_BONUS_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_BONUS_H

#include "Entity.h"

namespace Model {

class Bonus : public Entity
{
public:
        enum Sort
        {
                eJetpack = 0,
                eSpring = 1,
        };

private:
        Bonus::Sort mSort;

public:
        Model::Type getType() const override { return Model::Type::eBonus; }
        Sort getMSort() const;
        void setMSort(Sort sort);
};
} // namespace Model

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_BONUS_H
```

8.11 Entity.h

```
//
// Created by Pablo Deputter on 13/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_ENTITY_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_ENTITY_H

#include "Subject.h"
#include "util/Camera.h"
#include "util/Stopwatch.h"

#include "IVisitor.h"

#include <iostream>
#include <string>

namespace Model {
enum Type
{
        ePlayer = 0,
        eBonus = 1,
        eStatic = 2,
        eHorizontal = 3,
        eVertical = 4,
        eTemporary = 5,
        eBackground = 6,
        eJetpack = 7,
        eSpring = 8,
        eScore = 9
};

class CollisionBox
{
private:
        float mLeft;
        float mWidth;
        float mBottom;
        float mHeight;

public:
        CollisionBox() = default;

        CollisionBox(float left, float width, float bottom, float height)
            : mLeft(left), mWidth(width), mBottom(bottom), mHeight(height)
        {
        }

        ~CollisionBox() = default;

        [[nodiscard]] float getLeft() const { return mLeft; }

        [[nodiscard]] float getWidth() const { return mWidth; }

        [[nodiscard]] float getBottom() const { return mBottom; }

        [[nodiscard]] float getHeight() const { return mHeight; }
};

class Entity : public Observer::Subject, public Visitor::IVisitor
{
protected:
        float mX;
        float mY;
        //
        //
        float mWidth{};
        float mHeight{};
        bool mRemoveFlag;
        unsigned int mScore;

        float mSpawnRate;

public:
        Entity() : mX(0.f), mY(0.f), mWidth(0.f), mHeight(0.f), mRemoveFlag(false), mScore(0),
      mSpawnRate(0.f) {}

        Entity(unsigned int score, float spawnRate)
            : mX(0.f), mY(0.f), mWidth(0.f), mHeight(0.f), mRemoveFlag(false), mScore(score),
      mSpawnRate(spawnRate)
        {
        }

        virtual ~Entity() = default;
         float getX() const;
         float getY() const;
         void setX(float x);
         void setY(float y);


         float getWidth() const;
         float getHeight() const;
         void setWidth(float width);
         void setHeight(float height);
         virtual void move(bool collision) = 0;
         void move(float x, float y);
         virtual Model::Type getType() const = 0;
         virtual void onDestroy();

         void visit(Model::Player& player) override { std::cout « "Entity visit\n"; }

         virtual void accept(const std::shared_ptr<Visitor::IVisitor>& visitor) {}

         // TODO - jetpack
         void setRemoveFlag(bool flag)
         {
                 mRemoveFlag = flag;
                 onDestroy();
         }

         virtual bool getRemovable() const { return mRemoveFlag; }

         virtual bool isRemovable() const { return mRemoveFlag; }

         virtual bool isBonus() const { return false; }

         void setScore(unsigned int score) { mScore = score; }

         virtual unsigned int getScore() const { return mScore; }

         virtual float getSpawnRate() const { return mSpawnRate; }

         void setSpawnRate(float spawnRate) { mSpawnRate = spawnRate; }
 };
 } // namespace Model

 #endif // ADVANCED_PROGRAMMING_DOODLEJUMP_ENTITY_H
```

8.12 HorizontalPlatform.h

```
//
// Created by Pablo Deputter on 05/12/2021.
//

#ifndef DOODLEJUMP_HORIZONTALPLATFORM_H
#define DOODLEJUMP_HORIZONTALPLATFORM_H

#include "Entity.h"
#include "util/Random.h"

namespace Model {

class HorizontalPlatform : public Entity
{
private:
        std::pair<float, float> mBounds;
        bool mMovingForward;
        bool mInit;

public:
        HorizontalPlatform() : Entity(5, .10f), mBounds({0.f, 0.f}), mMovingForward(false), mInit(false)
      {}

        ~HorizontalPlatform() override = default;

        Model::Type getType() const override;

        void move(bool collision) override;

        void initBounds();
};
} // namespace Model

#endif // DOODLEJUMP_HORIZONTALPLATFORM_H
```

8.13 Jetpack.h

```
//
// Created by Pablo Deputter on 06/12/2021.
//

#ifndef DOODLEJUMP_JETPACK_H
#define DOODLEJUMP_JETPACK_H

#include "Player.h"

namespace Model {

class Jetpack : public Entity
{
private:
        std::pair<float, float> mBounds;
        bool mMovingDown;
        bool mInit;

public:
        bool mStarted;

public:
        Jetpack(bool started) : Entity(20, .25f), mMovingDown(false), mInit(false), mStarted(started) {}

        ~Jetpack() override = default;

        [[nodiscard]] Model::Type getType() const override;

        void move(bool collision) override;

        void initBounds();

        // TODO - jetpack
        void visit(Model::Player& player) override
        {
                std::cout « "visit\n";
                if (mStarted) {
                        std::cout « "return player state\n";
                        player.setDrag(0.006f);
                        return;
                }
                Utils::Stopwatch::getInstance().addTimer(Model::eJetpack, 2.5f);
                player.setDrag(0.f);
                mStarted = true;
        }
        // TODO - observer pattern mss
        // event met player dus dan kan die accepten als gedaan is
        bool isRemovable() const override
        {
                // moet removeflag en gestart zijn en de timer moet afgelopen zijn
                if (mStarted && mStarted && Utils::Stopwatch::getInstance().checkTimer(Model::eJetpack))
      {
                        std::shared_ptr<Model::Entity> jetpack = std::make_shared<Model::Jetpack>(true);

                        trigger(EventType::STOP_BONUS, std::make_shared<StopBonusEvent>(jetpack));
                        std::cout « "jetpackStopped\n";
                        return true;
                }
                return false;
        }

        bool isBonus() const override { return true; }
};
} // namespace Model

#endif // DOODLEJUMP_JETPACK_H
```

8.14 Platform.h

```
//
// Created by Pablo Deputter on 13/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORM_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORM_H

#include "Entity.h"
#include "util/Random.h"
#include <iostream>

namespace Model {

class Platform : public Entity
{
private:
        Model::Type mSort;

        std::pair<float, float> mBoundX = {0.f, 0.f};
        bool mMovingForward = true;

        std::pair<float, float> mBoundY = {0.f, 0.f};
        bool mMovingDown = true;

public:
        explicit Platform(Model::Type sort) : mSort(sort) {}

        Platform()
        {
                float rand = Utils::Random::getInstance().random(0.f, 1.f);
                // TODO - remove
                //              std::cout « rand « std::endl;
                if (rand <= .75f)
                        mSort = eStatic;
                else if (.85f >= rand >= .75f)
                        mSort = eHorizontal;
                else if (.95f >= rand >= .85f)
                        mSort = eVertical;
                else
                        mSort = eTemporary;
        }

        ~Platform() override = default;

        Model::Type getType() const override { return mSort; }

        void move(bool collision) override;

        void initBounds();
};
} // namespace Model

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORM_H
```

8.15 Player.h

```
//
// Created by Pablo Deputter on 14/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYER_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYER_H

#include "Entity.h"
#include "IVisitor.h"
#include <iostream>

namespace Model {
class Player : public Entity, public Observer::Observer, public IEventHandler
{
private:
        std::pair<float, float> mVelocity;
        std::pair<float, float> mDirection;
        const float mMaxVelocity = 0.20f;
        const float mAcceleration = 0.015f;
        float mDrag = 0.005f;
        bool mIsMovingLeft;
        bool mIsMovingRight;
public:
        Player() : mVelocity({0.f, 0.f}), mDirection({0.f, 0.f}), mIsMovingLeft(false),
    mIsMovingRight(false) {}
```

```

        ~Player() override = default;

        Model::Type getType() const override { return Model::Type::ePlayer; }

        void move(bool collision) override;

        const std::pair<float, float>& getVelocity() const;

        void setVelocity(const std::pair<float, float>& velocity);

        const std::pair<float, float>& getDirection() const;

        void setDirection(const std::pair<float, float>& direction);

        const float getMaxVelocity() const;

        const float getMaxAcceleration() const;

        const float getDrag() const;

        void setDrag(float drag);

        bool isMovingUp() const;

        void setIsMovingUp(bool isMovingUp);

        bool isMovingDown() const;

        void setIsMovingDown(bool isMovingDown);

        bool isMovingLeft() const;

        void setIsMovingLeft(bool isMovingLeft);

        bool isMovingRight() const;

        void setIsMovingRight(bool isMovingRight);

        // TODO - visitor pattern
        void accept(const std::shared_ptr<Visitor::IVisitor>& visitor) override { visitor->visit(*this);
    }

        void onTrigger(EventType type, const std::shared_ptr<Event>& event) override {
    event->send(*this); }

        void handleEvent(const StopBonusEvent& event) override
        {
                std::cout « "lol\n";
                accept(event.getBonus());
        }
};
} // namespace Model

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYER_H
```

8.16 Spring.h

```
//
// Created by Pablo Deputter on 06/12/2021.
//

#ifndef DOODLEJUMP_SPRING_H
#define DOODLEJUMP_SPRING_H

#include "Player.h"
#include <cmath>

namespace Model {

class Spring : public Entity
{
private:
        std::pair<float, float> mBounds;
        bool mMovingDown;
        bool mInit;

public:
        Spring() : Entity(15, .75f){};

        ~Spring() override = default;

        [[nodiscard]] Model::Type getType() const override;

        void move(bool collision) override;

        void initBounds();

        void visit(Model::Player& player) override
        {
                // Original jump with default speed
                float jumpPeak = (player.getMaxVelocity() / player.getDrag()) * (player.getMaxVelocity()
     / 2.f);
                // Calculate speed needed for a jump with a vertical distance x5
                float newSpeed = sqrt((jumpPeak * 5) * (player.getDrag() * 2.f));
                player.setVelocity({player.getVelocity().first, newSpeed});
        }

        bool isBonus() const override { return true; }

        bool isRemovable() const override { return mRemoveFlag; }
};
} // namespace Model

#endif // DOODLEJUMP_SPRING_H
```

8.17 StaticPlatform.h

```
//
// Created by Pablo Deputter on 05/12/2021.
//

#ifndef DOODLEJUMP_STATICPLATFORM_H
#define DOODLEJUMP_STATICPLATFORM_H

#include "Entity.h"

namespace Model {

class StaticPlatform : public Entity
{

public:
        StaticPlatform() : Entity(2, .70f) {}

        ~StaticPlatform() override = default;

        Model::Type getType() const override;

        void move(bool collision) override;
};
} // namespace Model

#endif // DOODLEJUMP_STATICPLATFORM_H
```

8.18 TemporaryPlatform.h

```
//
// Created by Pablo Deputter on 05/12/2021.
//

#ifndef DOODLEJUMP_TEMPORARYPLATFORM_H
#define DOODLEJUMP_TEMPORARYPLATFORM_H

#include "Entity.h"

namespace Model {

class TemporaryPlatform : public Entity
{

public:
        TemporaryPlatform() : Entity(10, 0.10f) {}

        ~TemporaryPlatform() override = default;

        Model::Type getType() const override;

        void move(bool collision) override;
};
} // namespace Model

#endif // DOODLEJUMP_TEMPORARYPLATFORM_H
```

8.19 VerticalPlatform.h

```
//
// Created by Pablo Deputter on 05/12/2021.
//

#ifndef DOODLEJUMP_VERTICALPLATFORM_H
#define DOODLEJUMP_VERTICALPLATFORM_H

#include "Entity.h"
#include "util/Random.h"

namespace Model {

class VerticalPlatform : public Entity
{
private:
        std::pair<float, float> mBounds;
        bool mMovingDown;
        bool mInit;

public:
        VerticalPlatform() : Entity(7, .10f), mBounds({0.f, 0.f}), mMovingDown(true), mInit(false) {}

        ~VerticalPlatform() override = default;

        Model::Type getType() const override;

        void move(bool collision) override;

        void initBounds();
};
} // namespace Model

#endif // DOODLEJUMP_VERTICALPLATFORM_H
```

8.20 Observer.h

```
//
// Created by Pablo Deputter on 14/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_OBSERVER_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_OBSERVER_H

#include "Event.h"

#include <memory>

namespace Observer {
// template<class EventType, class DataType>
class Observer
{
public:
        Observer() = default;
        virtual ~Observer() = default;
        virtual void onTrigger(EventType type, const std::shared_ptr<Event>& event) = 0;
};
} // namespace Observer

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_OBSERVER_H
```

8.21 Score.h

```
//
// Created by Pablo Deputter on 07/12/2021.
//

#ifndef DOODLEJUMP_SCORE_H
#define DOODLEJUMP_SCORE_H

#include "Event.h"
#include "model/Entity.h"
#include <Subject.h>

namespace Model {

class Score : public Entity, public Observer::Observer, IEventHandler
{
```

```
private:
        std::weak_ptr<Model::Entity> mLastCollision;

public:
        Score() = default;

        ~Score() override = default;

        [[nodiscard]] Model::Type getType() const override { return Model::eScore; }

        void setScore(unsigned int score) { Score::mScore += score; }

        void onTrigger(EventType type, const std::shared_ptr<Event>& event) override {
    event->send(*this); }

        void handleEvent(const NewMaxHeightEvent& event) override;

        void handleEvent(const CollisionEvent& event) override;

        void move(bool collision) override {}

        unsigned int getScore() const override { return mScore; }
};
} // namespace Model

#endif // DOODLEJUMP_SCORE_H
```

8.22 Settings.h

```
//
// Created by Pablo Deputter on 12/12/2021.
//

#ifndef DOODLEJUMP_SETTINGS_H
#define DOODLEJUMP_SETTINGS_H

namespace Settings {
static float CHANCE_STATIC = .9;
static float CHANCE_HORIZONTAL = .04;
static float CHANCE_VERTICAL = .04;
static float CHANCE_TEMPORARY = .02f;
static float CHANCE_BONUS = .1f;
static float CHANCE_SPRING = 1.;
static float CHANCE_JETPACK = .0f;
static unsigned int MIN_PLATFORMS = 7;
static unsigned int MAX_PLATFORMS = 20;
static float DIFFICULTY = 0.f;
enum Difficulty
{
        eEasy = 0,
        eNormal,
        eDifficult,
        eHard,
        eExtreme
};
bool static setDifficulty(Difficulty difficulty)
{
        switch (difficulty) {
        case eEasy:
                CHANCE_STATIC = .9f;
                CHANCE_HORIZONTAL = .04f;
                CHANCE_VERTICAL = .04f;
                CHANCE_TEMPORARY = .02f;

                CHANCE_BONUS = .05f;
                CHANCE_SPRING = .95f;
                CHANCE_JETPACK = .05f;

                MAX_PLATFORMS = 20;
                DIFFICULTY = 0.f;
                break;
        case eNormal:
                CHANCE_STATIC = .8f;
                CHANCE_HORIZONTAL = .07f;
                CHANCE_VERTICAL = .07f;
                CHANCE_TEMPORARY = .06f;

                CHANCE_BONUS = .15f;
                CHANCE_SPRING = .9f;
                CHANCE_JETPACK = .1f;

                MAX_PLATFORMS = 17;
                DIFFICULTY = 0.25f;
                break;
        case eDifficult:
                CHANCE_STATIC = .6f;
                CHANCE_HORIZONTAL = .15f;
                CHANCE_VERTICAL = .15f;
                CHANCE_TEMPORARY = .1f;

                CHANCE_BONUS = .25f;
                CHANCE_SPRING = .75;
                CHANCE_JETPACK = .25f;

                MAX_PLATFORMS = 14;
                DIFFICULTY = 0.5f;
                break;
        case eHard:
                CHANCE_STATIC = .4f;
                CHANCE_HORIZONTAL = .22f;
                CHANCE_VERTICAL = .22f;
                CHANCE_TEMPORARY = .16f;

                CHANCE_BONUS = .30f;
                CHANCE_SPRING = .75f;
                CHANCE_JETPACK = .25f;

                MAX_PLATFORMS = 10;
                DIFFICULTY = 0.75f;
                break;
        case eExtreme:
                CHANCE_STATIC = .2f;
                CHANCE_HORIZONTAL = .3f;
                CHANCE_VERTICAL = .3f;
                CHANCE_TEMPORARY = .2f;

                CHANCE_BONUS = .30f;
                CHANCE_SPRING = .75f;
                CHANCE_JETPACK = .25f;

                MAX_PLATFORMS = 7;
                DIFFICULTY = 0.85f;
                break;
        }
        return true;
}
} // namespace Settings

#endif // DOODLEJUMP_SETTINGS_H
```

8.23 Subject.h

```
//
// Created by Pablo Deputter on 14/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_SUBJECT_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_SUBJECT_H

#include "Observer.h"

#include <algorithm>
#include <memory>
#include <vector>

namespace Observer {
// template<class EventType, class Event>
class Subject
{
private:
        std::vector<std::shared_ptr<Observer» mObservers;
public:
        Subject() = default;
        virtual ~Subject() = default;
        void add(const std::shared_ptr<Observer>& observer) { mObservers.emplace_back(observer); }

        [[nodiscard]] const std::vector<std::shared_ptr<Observer»& getObservers() const { return
    mObservers; }
        void clear() { mObservers.clear(); }
        void trigger(EventType type, const std::shared_ptr<Event>& event) const
        {
                for (const auto& i : mObservers) {
                        i->onTrigger(type, event);
                }
        }
};
} // namespace Observer

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_SUBJECT_H
```

8.24 Camera.h

```
//
// Created by Pablo Deputter on 19/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_CAMERA_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_CAMERA_H

#include "Subject.h"

#include <memory>
#include <utility>

namespace Utils {
class Camera : public Observer::Subject
{
private:
        float mWorldLeft{};
        float mWorldRight{};
        float mWorldTop{};
        float mWorldBottom{};
        float mWindowLeft{};
        float mWindowRight{};
        float mWindowTop{};
        float mWindowBottom{};
        float mCameraX{};
        float mCameraY{};
        float mLastMaxHeight;
        float mMaxHeight;
        Camera() : mLastMaxHeight(0.f), mMaxHeight(0.f) {}

public:
        virtual ~Camera() = default;
        Camera(const Camera&) = delete;
        Camera& operator=(const Camera&) = delete;
        static Camera& getInstance();

        void reset();

        [[nodiscard]] std::pair<float, float> getWorldDimensions() const;
        void setWorldDimensions(float right, float top, float left = 0.f, float bottom = 0.f);
        [[nodiscard]] std::pair<float, float> getWindowDimensions() const;
        void setWindowDimensions(float right, float bottom, float left = 0.f, float top = 0.f);
        [[nodiscard]] std::pair<float, float> transform(float x, float y, float left = 0.f, float top =
      0.f) const;
        [[nodiscard]] std::pair<float, float> inverseTransform(float x, float y) const;
        void move(float x, float y);
        [[nodiscard]] float getX() const { return mCameraX; }
        [[nodiscard]] float getY() const { return mCameraY; }
        [[nodiscard]] float getMaxHeight() const { return mMaxHeight; }
        [[nodiscard]] float getLastMaxHeight() const { return mLastMaxHeight; }
        bool isMaxHeight(float height);
};
} // namespace Utils

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_CAMERA_H
```

8.25 Exception.h

```
//
// Created by Pablo Deputter on 09/12/2021.
//

#ifndef DOODLEJUMP_EXCEPTION_H
#define DOODLEJUMP_EXCEPTION_H

#include <exception>
#include <string>

namespace Utils {

class Exception : public std::exception
{
protected:
        std::string mValue;

public:
        explicit Exception(std::string value) : mValue(std::move(value)) {}

        Exception() = default;

        ~Exception() override = default;

        [[nodiscard]] const char* what() const noexcept override { return mValue.c_str(); }
};

class FileException : public Exception
{
public:
        explicit FileException(std::string file, std::string sort)
            : Exception("Failed to load " + std::move(sort) + " '" + std::move(file) + "'. File is
    missing.\n")
        {
        }
};
} // namespace Utils

#endif // DOODLEJUMP_EXCEPTION_H
```

8.26 Random.h

```
//
// Created by Pablo Deputter on 11/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_RANDOM_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_RANDOM_H

#include <random>

namespace Utils {
class Random
{
private:
        Random() = default;

public:
        ~Random() = default;
        Random(const Random&) = delete;
        Random& operator=(const Random&) = delete;
        static Random& getInstance();
        [[nodiscard]] float random(float a, float b);
};
} // namespace Utils

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_RANDOM_H
```

8.27 Stopwatch.h

```
//
// Created by Pablo Deputter on 14/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_STOPWATCH_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_STOPWATCH_H

#include "model/Entity.h"

#include <cfloat>
#include <chrono>
#include <iostream>
#include <map>

enum Type
{
        ePlayer = 0,
        eBonus = 1,
        eStatic = 2,
        eHorizontal = 3,
        eVertical = 4,
        eTemporary = 5,
        eBackground = 6,
        eJetpack = 7,
        eSpring = 8
};

namespace Utils {
class Stopwatch
{
private:
        std::chrono::high_resolution_clock::time_point mTime;
        float mDeltaTime{};
        std::map<Type, std::pair<float, std::chrono::high_resolution_clock::time_point» mTimers;

public:
        std::shared_ptr<Model::Entity> mPlayer;

        Stopwatch() = default;

public:
        ~Stopwatch() = default;
        Stopwatch(const Stopwatch&) = delete;
        Stopwatch& operator=(const Stopwatch&) = delete;
        static Stopwatch& getInstance();
        void start();
        [[nodiscard]] float lap();
        [[nodiscard]] float getDelta() const;

        // TODO - jetpack
        void addTimer(unsigned int key, float amount)
        {
                std::cout « "addTimer\n";
                if (mTimers.find(Type(key)) != std::end(mTimers)) {
                        //                        std::cout « "ERROR\n";
                }
                mTimers[Type(key)] = {amount, std::chrono::high_resolution_clock::now()};
        }
        //        std::chrono::duration<float> ms_delta = std::chrono::high_resolution_clock::now() -
    mTime;
        //        // Reset / lap stopwatch
        //        mTime = std::chrono::high_resolution_clock::now();
        //        mDeltaTime = ms_delta.count();
        //        // Return milliseconds as float
        //        return mDeltaTime;
        bool checkTimer(unsigned int key)
         {
                 if (mTimers.find(Type(key)) == std::end(mTimers)) {
                         //                        std::cout « "ERROR\n";
                         return false;
                 }
                 auto val = mTimers[Type(key)];
                 std::chrono::duration<float> ms_delta = std::chrono::high_resolution_clock::now() -
    val.second;
                 std::cout « ms_delta.count() « "\n";
                 if (ms_delta.count() > val.first) {
                         mTimers.erase(Type(key));
                         return false;
                 }
                 return true;
         }
};
} // namespace Utils

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_STOPWATCH_H
```

8.28 Utilities.h

```
//
// Created by Pablo Deputter on 21/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_COLLISION_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_COLLISION_H

#include "model/Entity.h"

#include <cmath>

namespace Utils {
class Utilities
{
public:
        static bool checkCollision(const std::shared_ptr<Model::Entity>& l, const
    std::shared_ptr<Model::Entity>& r);
        static bool checkWeight(float& rand, float weight);
};
} // namespace Utils

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_COLLISION_H
```

8.29 World.h

```
//
// Created by Pablo Deputter on 18/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_WORLD_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_WORLD_H

#include "controller/IController.h"
#include "controller/PlayerController.h"

#include "model/Entity.h"
#include "model/Player.h"

#include "AbstractFactory.h"

#include "model/Jetpack.h"

#include "HighScore.h"

#include "Score.h"
#include "util/Camera.h"
#include "util/Random.h"
#include "util/Utilities.h"

#include "Settings.h"

#include "Event.h"

#include <iostream>
#include <memory>
#include <vector>

class World
{
private:
        std::shared_ptr<Model::Player> mPlayer;
        std::vector<std::shared_ptr<Model::Entity» mEntities;
        std::shared_ptr<Model::AbstractFactory> mFactory;
        std::vector<std::shared_ptr<Model::Entity» mBackground;
        std::shared_ptr<Model::Score> mScore;
        unsigned int mActivePlatforms;
        Settings::Difficulty mDifficulty;

        bool mPlaying;

public:
        explicit World(std::shared_ptr<Model::AbstractFactory>& factory);

        ~World() { destroy(); }
        void initWorld();
        void events(const std::string& move, bool isPressed) const;
        void update();
        void render() const;
        void generateEntity();
        void spawnPlatform(float x, float y);
        void spawnBonus(float x, float y);
        void spawnEntity(float x, float y, Model::Type type);
        bool checkDifficulty();
        void addEntity(const std::shared_ptr<Model::Entity>& entity);
         void removeEntities();

         void destroy();

         [[nodiscard]] const std::shared_ptr<Model::Score>& getScore() const { return mScore; }

         [[nodiscard]] bool isPlaying() const { return mPlaying; }
};

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_WORLD_H
```

8.30 ConcreteFactory.h

```
//
```

```
// Created by Pablo Deputter on 21/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_CONCRETEFACTORY_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_CONCRETEFACTORY_H

#include "AbstractFactory.h"

#include "model/Background.h"
#include "model/HorizontalPlatform.h"
#include "model/Jetpack.h"
#include "model/Platform.h"
#include "model/Player.h"
#include "model/Spring.h"
#include "model/StaticPlatform.h"
#include "model/TemporaryPlatform.h"
#include "model/VerticalPlatform.h"

#include "controller/BonusController.h"
#include "controller/PlatformController.h"
#include "controller/PlayerController.h"

#include "view/BackgroundView.h"
#include "view/BonusView.h"
#include "view/IView.h"
#include "view/PlatformView.h"
#include "view/PlayerView.h"
#include "view/ScoreView.h"

#include "Score.h"

#include "SFML/Graphics.hpp"

// TODO - attach View Observers to Entity Subjects

namespace View {
class ConcreteFactory : public Model::AbstractFactory
{
public:
        ConcreteFactory() = default;

        ConcreteFactory(const std::shared_ptr<sf::RenderWindow>& window) : mWindow(window) {}

        ~ConcreteFactory() override = default;

        std::shared_ptr<Model::Player> createPlayer() override;

        std::shared_ptr<Model::Entity> createStaticPlatform() override;

        std::shared_ptr<Model::Entity> createHorizontalPlatform() override;

        std::shared_ptr<Model::Entity> createVerticalPlatform() override;

        std::shared_ptr<Model::Entity> createTemporaryPlatform() override;

        std::shared_ptr<Model::Entity> createSpring() override;

        std::shared_ptr<Model::Entity> createJetpack() override;

        std::shared_ptr<Model::Entity> createBackground() override;

        std::shared_ptr<Model::Score> createScore() override;

private:
        std::shared_ptr<sf::RenderWindow> mWindow;
};
} // namespace View

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_CONCRETEFACTORY_H
```

8.31 Game.h

```
//
// Created by Pablo Deputter on 14/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_GAME_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_GAME_H

#include "ConcreteFactory.h"

#include "World.h"

```

```
#include "util/Resourcemanager.h"
#include "util/Stopwatch.h"

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

class Game
{
        std::shared_ptr<sf::RenderWindow> mWindow;
        std::unique_ptr<World> mWorld;
        std::shared_ptr<Model::AbstractFactory> mFactory;
public:
        Game() = default;

        explicit Game(unsigned int width = 800, unsigned int height = 1440);

        ~Game() = default;

        void initializeResources();

        void processEvents();

        void handlePlayerInput(sf::Keyboard::Key key, bool isPressed);

        void render();

        void run();

        void drawHighScoreTable();
};

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_GAME_H
```

8.32 Resourcemanager.h

```
//
// Created by Pablo Deputter on 29/11/2021.
//

#ifndef DOODLEJUMP_RESOURCEMANAGER_H
#define DOODLEJUMP_RESOURCEMANAGER_H

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


#include "model/Entity.h"
#include "util/Exception.h"

#include <map>
#include <string>
#include <utility>
#include <typeinfo>

namespace Utils {
template<class Type>
struct Resourceholder
{
private:
        std::string mPath;
        std::map<Model::Type, std::shared_ptr<Type» mResources;
public:
        explicit Resourceholder(std::string path) : mPath(std::move(path)) {}

        ~Resourceholder() = default;

        void insert(Model::Type type, const std::string& subPath)
        {
                std::shared_ptr<Type> file = std::make_shared<Type>();
                std::string path = mPath + std::string(subPath);

                if (!file->loadFromFile(path)) {
                        throw(FileException(path, typeid(file).name()));
                }
                mResources.emplace(type, file);
        }

        std::shared_ptr<Type>& get(Model::Type type) { return mResources[type]; }
};

class Resourcemanager
{
private:
        std::shared_ptr<Resourceholder<sf::Texture» mTextures;
        std::shared_ptr<Resourceholder<sf::Font» mFonts;
        std::shared_ptr<Resourceholder<sf::SoundBuffer» mSounds;

        explicit Resourcemanager(const std::string& path)
        {
                mTextures = std::make_shared<Resourceholder<sf::Texture»(path);
                mFonts = std::make_shared<Resourceholder<sf::Font»(path);
                mSounds = std::make_shared<Resourceholder<sf::SoundBuffer»(path);
        }

public:
        ~Resourcemanager() = default;

        Resourcemanager(const Resourcemanager&) = delete;

        Resourcemanager& operator=(const Resourcemanager&) = delete;

        static Resourcemanager& getInstance()
        {
                static Resourcemanager instance(
                    "/Users/pablodeputter/Documents/GitHub/Advanced-Programming-DoodleJump/resource");
                return instance;
        }

        void addTexture(Model::Type type, const std::string& subPath) { mTextures->insert(type, subPath);
    }

        void addFont(Model::Type type, const std::string& subPath) { mFonts->insert(type, subPath); }

        void addSound(Model::Type type, const std::string& subPath) { mSounds->insert(type, subPath); }

        [[nodiscard]] const std::shared_ptr<Resourceholder<sf::Texture»& getTextures() const { return
    mTextures; }

        [[nodiscard]] const std::shared_ptr<Resourceholder<sf::Font»& getFonts() const { return mFonts; }

        [[nodiscard]] const std::shared_ptr<Resourceholder<sf::SoundBuffer»& getSounds() const { return
    mSounds; }
};
} // namespace Utils

#endif // DOODLEJUMP_RESOURCEMANAGER_H
```

8.33 BackgroundView.h

```
//
// Created by Pablo Deputter on 29/11/2021.
//

#ifndef DOODLEJUMP_BACKGROUNDVIEW_H
#define DOODLEJUMP_BACKGROUNDVIEW_H

#include "IView.h"

#include <filesystem>

namespace View {

class BackgroundView : public IView
{
public:
        BackgroundView(const std::shared_ptr<Model::Entity>& entity, const
    std::shared_ptr<sf::RenderWindow>& window)
            : IView(entity, window)
        {
                std::shared_ptr<sf::Texture>& tex =
                    Utils::Resourcemanager::getInstance().getTextures()->get(Model::eBackground);

                mSprite = std::make_unique<sf::Sprite>();
                mSprite->setTexture(*tex);
                mSprite->scale(1.f, 1.f);
                mSprite->setColor(sf::Color(255, 255, 255, 255));

                mEntity->setWidth((float)tex->getSize().x * mSprite->getScale().x);
                mEntity->setHeight((float)tex->getSize().y * mSprite->getScale().y);
        }

        ~BackgroundView() override = default;

        void handleEvent(const DrawEvent& event) override;

        void handleEvent(const OutOfViewEvent& event) override;
};
} // namespace View

#endif // DOODLEJUMP_BACKGROUNDVIEW_H
```

8.34 BonusView.h

```
//
// Created by Pablo Deputter on 06/12/2021.
//

#ifndef DOODLEJUMP_BONUSVIEW_H
#define DOODLEJUMP_BONUSVIEW_H

#include "IView.h"

namespace View {

class BonusView : public IView
{
public:
        BonusView(const std::shared_ptr<Model::Entity>& entity, const std::shared_ptr<sf::RenderWindow>&
     window)
              : IView(entity, window)
        {
                Model::Type type = entity->getType();
                std::shared_ptr<sf::Texture>& tex =
     Utils::Resourcemanager::getInstance().getTextures()->get(type);

                mSprite = std::make_unique<sf::Sprite>();
                mSprite->setTexture(*tex);
                mSprite->scale(.1, .1f);

                auto texSize = Utils::Camera::getInstance().inverseTransform(
                    (float)tex->getSize().x * mSprite->getScale().x, (float)tex->getSize().y *
     mSprite->getScale().y);

                mEntity->setWidth(texSize.first);
                mEntity->setHeight(Utils::Camera::getInstance().getWorldDimensions().second -
     texSize.second);
        }

        BonusView() = default;

        ~BonusView() override = default;
};
} // namespace View

#endif // DOODLEJUMP_BONUSVIEW_H
```

8.35 IView.h

```
//
// Created by Pablo Deputter on 19/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_IVIEW_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_IVIEW_H

#include "model/Bonus.h"
#include "model/Player.h"

#include "Observer.h"
#include "util/Camera.h"

#include "util/Resourcemanager.h"

#include "SFML/Graphics.hpp"

#include "Event.h"

#include <iostream>
#include <memory>

namespace View {

class IView : public Observer::Observer, public IEventHandler, public
     std::enable_shared_from_this<Observer::Observer>
{
```

```
protected:
        std::shared_ptr<Model::Entity> mEntity;
        std::unique_ptr<sf::Sprite> mSprite;
        std::shared_ptr<sf::RenderWindow> mWindow;
        std::unique_ptr<sf::Sound> mSound;

public:
        IView(const std::shared_ptr<Model::Entity>& entity, const std::shared_ptr<sf::RenderWindow>&
    window);

        IView() = default;

        virtual ~IView() = default;

        void drawCollisionBox();

        void onTrigger(EventType type, const std::shared_ptr<Event>& event) override;

        void handleEvent(const DrawEvent& event) override;

        void handleEvent(const OutOfViewEvent& event) override;

        virtual void handleEvent(const CollisionEvent& event) override {}

        void handleEvent(const NewDifficultyEvent& event) override {}

        template <class Type>
        static void setRainbowColor(const std::unique_ptr<sf::Text>& object)
        {
                if (object->getFillColor().r + 5 <= 255 && object->getFillColor().g == 0 &&
                    object->getFillColor().b == 0) {
                        object->setFillColor(sf::Color(object->getFillColor().r + 5, 0, 0));
                } else if (object->getFillColor().r == 255 && object->getFillColor().g + 5 <= 255 &&
                           object->getFillColor().b == 0) {
                        object->setFillColor(sf::Color(255, object->getFillColor().g + 5, 0));
                } else if (object->getFillColor().r - 5 >= 0 && object->getFillColor().g == 255 &&
                           object->getFillColor().b == 0) {
                        object->setFillColor(sf::Color(object->getFillColor().r - 5, 255, 0));
                } else if (object->getFillColor().r == 0 && object->getFillColor().g == 255 &&
                           object->getFillColor().b + 5 <= 255) {
                        object->setFillColor(sf::Color(0, 255, object->getFillColor().b + 5));
                } else if (object->getFillColor().r == 0 && object->getFillColor().g - 5 >= 0 &&
                           object->getFillColor().b == 255) {
                        object->setFillColor(sf::Color(0, object->getFillColor().g - 5, 255));
                } else if (object->getFillColor().r + 5 <= 255 && object->getFillColor().g == 0 &&
                           object->getFillColor().b == 255) {
                        object->setFillColor(sf::Color(object->getFillColor().r + 5, 0, 255));
                } else if (object->getFillColor().r == 255 && object->getFillColor().g == 0 &&
                           object->getFillColor().b - 5 >= 0) {
                        object->setFillColor(sf::Color(255, 0, object->getFillColor().b - 5));
                }
        }
};
} // namespace View

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_IVIEW_H
```

8.36 PlatformView.h

```
//
// Created by Pablo Deputter on 19/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORMVIEW_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORMVIEW_H

#include "IView.h"
#include <thread>

namespace View {

class PlatformView : public IView
{
public:
        PlatformView(const std::shared_ptr<Model::Entity>& entity, const
    std::shared_ptr<sf::RenderWindow>& window)
            : IView(entity, window)
        {
                Model::Type type = entity->getType();
                std::shared_ptr<sf::Texture>& tex =
    Utils::Resourcemanager::getInstance().getTextures()->get(type);

                mSprite->setTexture(*tex);
                mSprite->scale(.25f, .25f);

                // TODO - sound
                //
mSound->setBuffer(*Utils::Resourcemanager::getInstance().getSounds()->get(mEntity->getType()));
                //                mSound->setVolume(100.f);

                auto texSize = Utils::Camera::getInstance().inverseTransform(
                    (float)tex->getSize().x * mSprite->getScale().x, (float)tex->getSize().y *
mSprite->getScale().y);

                mEntity->setWidth(texSize.first);
                mEntity->setHeight(Utils::Camera::getInstance().getWorldDimensions().second -
texSize.second);
        }

        PlatformView() = default;

        ~PlatformView() override = default;

        void handleEvent(const CollisionEvent& event) override
        {
                // TODO - threads leaks
                //                mSound->play();
        }
};
} // namespace View

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_PLATFORMVIEW_H
```

8.37 PlayerView.h

```
//
// Created by Pablo Deputter on 19/11/2021.
//

#ifndef ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYERVIEW_H
#define ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYERVIEW_H

#include "IView.h"

namespace View {

class PlayerView : public IView
{
public:
        PlayerView(const std::shared_ptr<Model::Entity>& entity, const std::shared_ptr<sf::RenderWindow>&
window)
            : IView(entity, window)
        {

                std::shared_ptr<sf::Texture>& tex =
                    Utils::Resourcemanager::getInstance().getTextures()->get(Model::ePlayer);

                mSprite = std::make_unique<sf::Sprite>();
                mSprite->setTexture(*tex);
                mSprite->scale(2.f, 2.f);

                auto texSize = Utils::Camera::getInstance().inverseTransform(
                    (float)tex->getSize().x * mSprite->getScale().x, (float)tex->getSize().y *
mSprite->getScale().y);

                mEntity->setWidth(texSize.first);
                mEntity->setHeight(Utils::Camera::getInstance().getWorldDimensions().second -
texSize.second);
        }

        PlayerView() = default;

        ~PlayerView() override = default;
};
} // namespace View

#endif // ADVANCED_PROGRAMMING_DOODLEJUMP_PLAYERVIEW_H
```

8.38 ScoreView.h

```
//
// Created by Pablo Deputter on 09/12/2021.
```

```cpp
//

#ifndef DOODLEJUMP_SCOREVIEW_H
#define DOODLEJUMP_SCOREVIEW_H

#include "IView.h"

#include "Score.h"

namespace View {

class ScoreView : public IView
{
private:
        std::unique_ptr<sf::Text> mText;
        std::unique_ptr<sf::Text> mDiffText;

        void handleEvent(const DrawEvent& event) override;

        void handleEvent(const NewDifficultyEvent& event) override;

public:
        ScoreView(std::shared_ptr<Model::Score>& entity, const std::shared_ptr<sf::RenderWindow>& window)
            : IView(entity, window)
        {
                std::shared_ptr<sf::Font>& font =
      Utils::Resourcemanager::getInstance().getFonts()->get(Model::eScore);

                mSprite = std::make_unique<sf::Sprite>();
                mText = std::make_unique<sf::Text>();
                mText->setFont(*font);
                mText->setFillColor(sf::Color::Black);

                mDiffText = std::make_unique<sf::Text>();
                mDiffText->setFont(*font);
                mDiffText->setFillColor(sf::Color::Black);
        }

        ~ScoreView() override = default;
};
} // namespace View

#endif // DOODLEJUMP_SCOREVIEW_H
```

Index