WFCPP

0.0.1

Generated by Doxygen 1.9.5

Chapter 1

Concept Index

1.1 Concepts

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Concept Documentation

4.1 PixelType Concept Reference

The concept for pixel-like objects. It requires the type to contain RGBA properties/members.

`#include <Pixel.h>`

4.1.1 Concept definition

```
template<typename T>
concept PixelType =  requires
{
    T::Red;
    T::Green;
    T::Blue;
    T::Alpha;
}
```

4.1.2 Detailed Description

The concept for pixel-like objects. It requires the type to contain RGBA properties/members.

Template Parameters

T	

Chapter 5

Class Documentation

5.1 BMPImage Class Reference

The BMPImage class.

`#include <BMPImage.h>`

Public Member Functions

- BMPImage ()
  Construct a new BMPImage object.
- BMPImage (unsigned int initWidth, unsigned int initHeight)
  Construct a new BMPImage object.
- BMPImage (std::string filename)
  Construct a new BMPImage object from image file.
- ~BMPImage ()=default
  Destroy the BMPImage object.
- unsigned int getWidth () const noexcept
  Get the width.
- unsigned int getHeight () const noexcept
  Get the height.
- void setSize (unsigned int NewWidth, unsigned int NewHeight)
  Set the Size of the image. The added positions will be filled with defaultPixel.
- Pixel getPixel (Position pos) const
  Get the Pixel object at some position.
- void setPixel (Position pos, Pixel newPixel)
  Set the Pixel object at some position.
- void exportToFile (std::string filename, FileType type) const
  Export this object to a file of specified type.

5.1.1 Detailed Description

The BMPImage class.

This class supports the import, export and basic modification of images as BMP files. It provides an interface for image processing in Extractor and Synthesizer.

At the time, we only support png and bmp format images.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BMPImage() [1/3]

```
BMPImage::BMPImage ( )
```

Construct a new BMPImage object.

5.1.2.2 BMPImage() [2/3]

```
BMPImage::BMPImage (
            unsigned int initWidth,
            unsigned int initHeight )
```

Construct a new BMPImage object.

Parameters

initWidth	
initHeight	

5.1.2.3 BMPImage() [3/3]

```
BMPImage::BMPImage (
            std::string filename )
```

Construct a new BMPImage object from image file.

Parameters

filename	File to read from.

5.1.2.4 ~BMPImage()

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

Destroy the BMPImage object.

5.1.3 Member Function Documentation

5.1.3.1 exportToFile()

```
void BMPImage::exportToFile (
            std::string filename,
            FileType type ) const
```

Export this object to a file of specified type.

Parameters

filename	The path for the exported image.
type	The type of the image.

Exceptions

std::runtime_error	Indicates that the export fails.

5.1.3.2 getHeight()

```
unsigned int BMPImage::getHeight ( ) const  [noexcept]
```

Get the height.

Returns

the height as unsigned int

5.1.3.3 getPixel()

```
Pixel BMPImage::getPixel (
            Position pos ) const
```

Get the Pixel object at some position.

Parameters

pos	a position in image

Returns

Pixel

5.1.3.4 getWidth()

```
unsigned int BMPImage::getWidth ( ) const  [noexcept]
```

Get the width.

Returns

the width as unsigned int

5.1.3.5 setPixel()

```
void BMPImage::setPixel (
            Position pos,
            Pixel newPixel )
```

Set the Pixel object at some position.

Parameters

pos	a position in image
newPixel	the new Pixel to be filled in

Exceptions

std::out_of_range	Indicates that the position is not valid.

5.1.3.6 setSize()

```
void BMPImage::setSize (
            unsigned int NewWidth,
            unsigned int NewHeight )
```

Set the Size of the image. The added positions will be filled with defaultPixel.

Parameters

NewWidth	
NewHeight	

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

- include/BMPImage.h
- src/BMPImage.cpp

5.2 Extractor Class Reference

The Extractor Class.

`#include <Extractor.h>`

Public Member Functions

- void extractPNG (unsigned int &width, unsigned int &height, const std::string &filename, std::vector< Pixel > &pixels)
- void extractBMP (unsigned int &width, unsigned int &height, const std::string &filename, std::vector< Pixel > &pixels)
- int extractTileset (unsigned int width, unsigned int height, unsigned int horizontal, unsigned int vertical, const std::vector< Pixel > &pixels, std::vector< TileData > &tiles)
- void encodeTilesetPNG (std::vector< TileData > &tiles, std::string assetName)
- int setConstraints (std::vector< TileData > &tileList)
- void setComparisonMetric (double m)
- void setDumpDirectoryRelative (std::string dir)
- void setDumpDirectoryAbsolute (std::string dir)
- void setAssetDirectoryRelative (std::string dir)
- void setAssetDirectoryAbsolute (std::string dir)
- void printDirectories ()
- void printConstraintsV (std::vector< TileData > tiles)
- void printConstraints (TileData tile)
- Extractor ()
- Extractor (double metric)

Static Public Member Functions

- static void encodePNG (unsigned int width, unsigned int height, const std::string &filename, const std::vector< Pixel > &pixels)

5.2.1 Detailed Description

The Extractor Class.

Author

Ricardo Figueroa

This class provides support for the Wave Function Collapse algorithm by making available png and bitmap extraction and encoding for tiles and tilesets. Tilesets are usually formatted in a X by X image file, so the extractor takes said image file and extracts all necessary information, i.e, pixel data, width and height.

There existed many methods by which to expand the extractor class. As a result of time constraints png and bmp are the only filetypes available for extraction. But by introducing more libraries to the class it can be possible to expand to a larger range of filetypes, that said, the most common ones have been chosen for the class.

Using both lodePNG and EasyBMP, extraction and encoding was made much easier. Allowing for simple to use API that abstract these processes.

Extraction for both PNG and BMP consists of acquiring the RGBA values of each pixel. Ranging from 0 to 255 for each. They are placed in a vector of Pixel type that is meant to be the generalized structure for all extractions (space for appending functionality)

Encoding for both PNG and BMP consists of taking this generalized structure and encoding through their respective support libraries.

The vectors of pixels as described above are then processed by extractTileset, which is the function that creates the required piece for both the Solver and Synthesizer (a vector of TileData). Given that the pixels themselves aren't organized in Tile form.

Once the TileData is extracted it can then be automatically be setConstraints based on a comparison metric. This is highly basic at the moment, the metric being a int meant to represent a percentage from 0 to 1.0. The difference in tile sides is calculated and compared to this metric to decide whether the tiles are accepted as constraints. This can be expanded on by changing the metric to some AI driven metric. Knowing what differernce in pixels is acceptable is hard to measure through simple arithmatic so for now it is just an option availble to the user.

See also

Pixel

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Extractor() [1/2]

```
Extractor::Extractor ( )  [inline]
```

Default constructor for the Extractor.

5.2.2.2 Extractor() [2/2]

```
Extractor::Extractor (
            double metric )  [inline]
```

Constructor for the Extractor, allows for metric initialization.

Parameters

metric	The metric the user is allowed to implement. Max at 1, min at 0

5.2.3 Member Function Documentation

5.2.3.1 encodePNG()

```
void Extractor::encodePNG (
            unsigned int width,
            unsigned int height,
            const std::string & filename,
            const std::vector< Pixel > & pixels )  [static]
```

Encodes a vector of pixels into a PNG file at the filename provided.

Parameters

width	The width of the file being created
height	The height of the file being created
filename	The filename of the file being created
pixels	The pixels used to created the file

5.2.3.2 encodeTilesetPNG()

```
void Extractor::encodeTilesetPNG (
            std::vector< TileData > & tiles,
            std::string assetName )
```

Econdes a vecotr of tiles using encodePNG

Parameters

tiles	The vector of tiles being encoded
assetName	name of asset and folder

5.2.3.3 extractBMP()

```
void Extractor::extractBMP (
            unsigned int & width,
```

```
            unsigned int & height,
            const std::string & filename,
            std::vector< Pixel > & pixels )
```

Extracts the pixel data from a BMP file.

Parameters

width	A reference to width that gets updated on easyBMP extraction
height	A reference to height that gets updated on easyBMP extraction
filename	A reference to filename that is used to retrieve the desired file
pixels	The pixel data of a file

See also

Pixel

5.2.3.4 extractPNG()

```
void Extractor::extractPNG (
            unsigned int & width,
            unsigned int & height,
            const std::string & filename,
            std::vector< Pixel > & pixels )
```

Extracts the pixel data from a PNG file.

Parameters

width	A reference to width that gets updated on lodePNG extraction
height	A reference to height that gets updated on lodePNG extraction
filename	A reference to filename that is used to retrieve the desired file
pixels	The pixel data of a file

See also

Pixel

5.2.3.5 extractTileset()

```
int Extractor::extractTileset (
            unsigned int width,
            unsigned int height,
            unsigned int horizontal,
            unsigned int vertical,
```

```
const std::vector< Pixel > & pixels,
std::vector< TileData > & tiles )
```

Extracts tiles from a vector of pixels.

Parameters

width	The width of the file that extracted the pixels referenced in
height	The height of the file that extracted the pixels referenced in
horizontal	The maximum number of tiles on the tileset horizontally
vertical	The maxumum number of tiles on the tilset vertically
pixels	A reference to the pixels that will be extracted
tiles	A reference to the tile vector that will be appended to

5.2.3.6 printConstraints()

```
void Extractor::printConstraints (
            TileData tile )
```

Print function for constraints of a tile

Parameters

tile	Tile for printing

5.2.3.7 printConstraintsV()

```
void Extractor::printConstraintsV (
            std::vector< TileData > tiles )
```

Print function for constraints of a vector of tiles

Parameters

tile	TileVector for printing

5.2.3.8 printDirectories()

```
void Extractor::printDirectories ( )
```

Print function for the two directories

5.2.3.9 setAssetDirectoryAbsolute()

```
void Extractor::setAssetDirectoryAbsolute (
            std::string dir )
```

Directory setter for acquiring assets

Parameters

dir	Directory

5.2.3.10 setAssetDirectoryRelative()

```
void Extractor::setAssetDirectoryRelative (
            std::string dir )
```

Directory setter for acquiring assets relative to current directory

Parameters

dir	Directory

5.2.3.11 setComparisonMetric()

```
void Extractor::setComparisonMetric (
            double m )
```

Setter function for comparisonMetric

Parameters

m	Input for comparisonMetric

5.2.3.12 setConstraints()

```
int Extractor::setConstraints (
            std::vector< TileData > & tileList )
```

Set the constraints for each tile on a tileList based on the tiles from that tileList.

Parameters

tileList	A vector of tiles to apply constraints to

5.2.3.13 setDumpDirectoryAbsolute()

```
void Extractor::setDumpDirectoryAbsolute (
            std::string dir )
```

Directory setter for dumping assets

Parameters

dir	Directory

5.2.3.14 setDumpDirectoryRelative()

```
void Extractor::setDumpDirectoryRelative (
            std::string dir )
```

Directory setter for dumping extractions relative to current directory

Parameters

dir	Directory

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

- include/Extractor.h
- src/Extractor.cpp

5.3 Grid< TileKey > Class Template Reference

Public Member Functions

- **Grid** (size_t dimension, std::map< TileKey, std::shared_ptr< Tile > > map)
- **Grid** (size_t dimension)
- size_t **getDimension** () const
- void **setDimension** (size_t newDimension)
- TileKey **getKey** (Position p)
- void **setKey** (Position p, TileKey key)
- std::shared_ptr< Tile > **getTile** (Position pos) const
- void **setTile** (Position pos, TileKey tileKey)
- void **setTileMap** (std::map< TileKey, std::shared_ptr< Tile > > newMap)
- Position **translatePixelPosition** (Position pos) const
- std::vector< Position > **enumeratePosition** () const

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

- include/Grid.h

5.4 std::hash< Position > Struct Reference

Public Member Functions

- std::size_t **operator()** (const Position &k) const

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

- include/Solver.h

5.5 std::hash< std::pair< size_t, Direction > > Struct Reference

Public Member Functions

- std::size_t **operator()** (const std::pair< size_t, Direction > &k) const

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

- include/Solver.h

5.6 Pixel Struct Reference

The struct for storing pixels in BMP. Supports up to 32-bit colors.

`#include <Pixel.h>`

Public Member Functions

- template<PixelType T>
  Pixel & operator= (const T &rhs)
  Overload the = operator to support assignment from pixel-like objects.

Public Attributes

- int **count**
- BYTE Blue
  Blue value.
- BYTE Green
  Green value.
- BYTE Red
  Red value.
- BYTE Alpha
  Alpha value.

5.6.1 Detailed Description

The struct for storing pixels in BMP. Supports up to 32-bit colors.

5.6.2 Member Function Documentation

5.6.2.1 operator=()

```
template<PixelType T>
Pixel & Pixel::operator= (
            const T & rhs )  [inline]
```

Overload the = operator to support assignment from pixel-like objects.

Template Parameters

T	

Parameters

rhs	A pixel-like object

Returns

Pixel&

See also

PixelType

5.6.3 Member Data Documentation

5.6.3.1 Alpha

```
BYTE Pixel::Alpha
```

Alpha value.

5.6.3.2 Blue

```
BYTE Pixel::Blue
```

Blue value.

5.6.3.3 Green

```
BYTE Pixel::Green
```

Green value.

5.6.3.4 Red

```
BYTE Pixel::Red
```

Red value.

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

- include/Pixel.h

5.7 Position Struct Reference

The struct for storing grid coordinates. Overloads == operator in the usual manner.

```
#include <Position.h>
```

Public Member Functions

- Direction getDirection (Position other)

  A method for determining to what direction of the current Position any other Position is.

Public Attributes

- size_t **x**
- size_t **y**

5.7.1 Detailed Description

The struct for storing grid coordinates. Overloads == operator in the usual manner.

5.7.2 Member Function Documentation

5.7.2.1 getDirection()

```
Direction Position::getDirection (
            Position other )  [inline]
```

A method for determining to what direction of the current Position any other Position is.

Parameters

other	The other position

Returns

the direction from the current position that the other position is located in

See also

Direction

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

- include/Position.h

5.8 Solver Class Reference

The Solver class.

`#include <Solver.h>`

Public Types

- typedef size_t TileKey
- typedef std::function< std::vector< TileKey >::const_iterator(const std::vector< TileKey > &)> **CollapseBehavior**
- typedef std::function< void(const TileKey &, Position)> **CollapseCallback**
- typedef std::function< void(const std::vector< TileKey > &, Position)> **PropagateCallback**
- typedef std::list< CollapseCallback >::iterator CollapseCallbackCookie
- typedef std::list< PropagateCallback >::iterator PropagateCallbackCookie

Public Member Functions

- template<typename T >
  Solver (const std::vector< T > &tiles, int seed=0)
- template<typename T >
  Solver (const std::map< TileKey, T > &tiles, int seed=0)
- void setSeed (int seed)

  Set the seed for the random number generator.
- int getSeed ()

  Get the seed for the random number generator.
- void solve (size_t N, Grid< TileKey > &grid)
- void addAdjacencyConstraint (TileKey t, Direction d, TileKey neighbor)
- void removeAdjacencyConstraint (TileKey t, Direction d, TileKey neighbor)
- void addAdjacencyConstraint (TileKey t, Direction d, std::initializer_list< TileKey > neighbors)
- void removeAdjacencyConstraint (TileKey t, Direction d, std::initializer_list< TileKey > neighbors)
- void setInitialConstraint (Position p, TileKey possibility)
- void setInitialConstraint (Position p, std::initializer_list< TileKey > possibilities)
- CollapseCallbackCookie registerOnCollapse (CollapseCallback callback)
- void deregisterOnCollapse (CollapseCallbackCookie cookie)
- PropagateCallbackCookie registerOnPropagate (PropagateCallback callback)
- void deregisterOnCollapse (PropagateCallbackCookie cookie)
- void setCollapseBehaviour (std::optional< CollapseBehavior > b)

5.8.1 Detailed Description

The Solver class.

Author

Jose A. Ramos

This class implements the Wave Function Collapse (WFC) algorithm. It provides an interface for calling the algorithm, for setting the solver constraints, changing some algorithm behavior, and registering for certain events. It utilizes keys to represent tiles, returning a 2D grid of keys.

At the moment, only 2 dimensional square grids are available to be solved by the algorithm. This limitation is not hard to overcome, and in fact this class can be generalized with minor changes to some key private methods and the Direction enum.

The algorithm consists of a grid, with each slot/square having a set of allowed tiles at any given time. In the WFC literature, this is called a "superposition", although we avoid that nomenclature. A grid square is said to be "collapsed" when only one tile is allowed in it. The grid itself is said to be "collapsed" when all of its squares are collapsed. This constitutes a "solved" grid. If a grid square ever has zero allowed tiles, that means the grid is in "contradiction". A solution cannot be found from a contradiction.

WFC works by iterating 3 steps until either the grid is solved or a leads to a contradiction:

- Find the non-collapsed grid square with the least number of allowed tiles (this is the "min entropy heuristic").
- Collapse the grid square using some policy (usually uniform random sampling).
- Propagate the results of the new collapsed grid square, removing tiles that are now disallowed by adjacency constraints.

Beyond offering the solve method, this class provides an API to define *adjacency constraints* and *initial constraints*, which guide the final solution to the grid. Additionally, it provides an API for registering functions to be called on the information at a grid square whenever it either collapses or is affected by propagation. Finally, an API is provided for changing the way grid squares are collapsed.

See also

Grid

Direction

Position

5.8.2 Member Typedef Documentation

5.8.2.1 CollapseCallbackCookie

```
typedef std::list<CollapseCallback>::iterator Solver::CollapseCallbackCookie
```

The alias for the callback cookie type for collapse events

See also

registerOnCollapse(CollapseCallback callback)

5.8.2.2 PropagateCallbackCookie

```
typedef std::list<PropagateCallback>::iterator Solver::PropagateCallbackCookie
```

The alias for the callback cookie type for propagation events

See also

registerOnPropagate(PropagateCallback callback)

5.8.2.3 TileKey

```
typedef size_t Solver::TileKey
```

The alias for the key type used for tiles in the algorithm.

5.8.3 Constructor & Destructor Documentation

5.8.3.1 Solver() [1/2]

```
template<typename T >
Solver::Solver (
            const std::vector< T > & tiles,
            int seed = 0 )  [inline]
```

The tile keys are auto-generated from vector as the indeces.

Parameters

tiles	A vector of tiles.
seed	The seed for the random number generator.

5.8.3.2 Solver() [2/2]

```
template<typename T >
Solver::Solver (
            const std::map< TileKey, T > & tiles,
            int seed = 0 )  [inline]
```

Keeps track of tile keys from a given mapping of tile keys to tiles.

Parameters

tiles	A map of tile keys to tiles.
seed	The seed for the random number generator.

5.8.4 Member Function Documentation

5.8.4.1 addAdjacencyConstraint() [1/2]

```
void Solver::addAdjacencyConstraint (
            TileKey t,
            Direction d,
            std::initializer_list< TileKey > neighbors )
```

A multi-argument version of Solver::addAdjacencyConstraint Utilizes an initializer list, for use by programmers.

This method is idempotent.

See also

Solver::addAdjacencyConstraint(TileKey t, Direction d, TileKey neighbor)

Parameters

t	The tile key which will have a new possible neighbor
d	The direction in which the neighbor will be possible
neighbors	The tile keys which are allowed to be adjacent to t in direction d

5.8.4.2 addAdjacencyConstraint() [2/2]

```
void Solver::addAdjacencyConstraint (
            TileKey t,
            Direction d,
            TileKey neighbor )
```

A method for adding an *adjacency constraint*. An adjacency constraint consists of a given tile, a direction, and a neighbor tile. The constraint specifies that the neighbor tile is allowed to be adjacent to the given tile in said direction. Inversely, if such a constraint does not exist, the neighbor tile is not allowed to be adjacent in that direction.

By default, if **no** constraint exists for a (tile, direction) pair, every tile is allowed to be adjacent to the given tile in that direction. This is why they are called *constraints*, since it constrains this default case by only allowing specific neighbors.

This method is idempotent.

Parameters

t	The tile key which will have a new possible neighbor
d	The direction in which the neighbor will be possible
neighbor	The tile key which is allowed to be adjacent to t in direction d

5.8.4.3 deregisterOnCollapse() [1/2]

```
void Solver::deregisterOnCollapse (
            Solver::CollapseCallbackCookie cookie )
```

Deregisters a function that is called whenever a grid square collapses.

See also

CollapseCallbackCookie

registerOnCollapse(CollapseCallback callback)

Parameters

cookie	a cookie used to indentify and remove a callback

5.8.4.4 deregisterOnCollapse() [2/2]

```
void Solver::deregisterOnCollapse (
            Solver::PropagateCallbackCookie cookie )
```

Deregisters a function that is called whenever a grid square is interacted with during constraint propagation.

See also

PropagateCallbackCookie

registerOnPropagate(PropagateCallback callback)

Parameters

cookie	a cookie used to indentify and remove a callback

5.8.4.5 getSeed()

```
int Solver::getSeed ( )
```

Get the seed for the random number generator.

Returns

the seed

5.8.4.6 registerOnCollapse()

```
Solver::CollapseCallbackCookie Solver::registerOnCollapse (
            Solver::CollapseCallback callback )
```

Registers a function to be called whenever a grid square collapses. The function argument is the tile key now occupying that grid square, and the position. Returns a cookie to allow deregistering the function later.

See also

CollapseCallback

CollapseCallbackCookie

deregisterOnCollapse(CollapseCallbackCookie cookie)

Parameters

callback	a function to be called when a grid square collapses

Returns

a cookie to be used for deregistering the callback

5.8.4.7 registerOnPropagate()

```
Solver::PropagateCallbackCookie Solver::registerOnPropagate (
            Solver::PropagateCallback callback )
```

Registers a function to be called whenever a grid square collapses. The function argument is the tile key now occupying that grid square, and the position. Returns a cookie to allow deregistering the function later.

See also

PropagateCallback

PropagateCallbackCookie

deregisterOnCollapse(PropagateCallbackCookie cookie)

Parameters

callback	a function to be called when a grid square is interacted with during constraint propagation.

Returns

a cookie to be used for deregistering the callback

5.8.4.8 removeAdjacencyConstraint() [1/2]

```
void Solver::removeAdjacencyConstraint (
            TileKey t,
            Direction d,
            std::initializer_list< TileKey > neighbors )
```

A multi-argument version of Solver::addAdjacencyConstraint Utilizes an initializer list, for use by programmers.

This method is idempotent.

See also

Solver::removeAdjacencyConstraint(TileKey t, Direction d, TileKey neighbor)

Parameters

t	The tile key which will have a possible neighbor removed
d	The direction in which the neighbor will no longer be a possible
neighbors	The tile keys which are no longer allowed to be adjacent to t in direction d

5.8.4.9 removeAdjacencyConstraint() [2/2]

```
void Solver::removeAdjacencyConstraint (
            TileKey t,
            Direction d,
            TileKey neighbor )
```

A method for removing an *adjacency constraint*.

This method is idempotent.

See also

Solver::addAdjacencyConstraint(TileKey t, Direction d, TileKey neighbor)

Parameters

t	The tile key which will have a possible neighbor removed
d	The direction in which the neighbor will no longer be a possible
neighbor	The tile key which is no longer allowed to be adjacent to t in direction d

5.8.4.10 setCollapseBehaviour()

```
void Solver::setCollapseBehaviour (
            std::optional< CollapseBehavior > b )
```

Low level access function to change selection policy used in collapse. Can be set to a null value, in which case collapseRandom is used instead.

See also

CollapseBehavior

Parameters

b	a function which returns an iterator to a specific tile key in a grid square

5.8.4.11 setInitialConstraint() [1/2]

```
void Solver::setInitialConstraint (
            Position p,
            std::initializer_list< TileKey > possibilities )
```

A multi-tile version of setInitialConstraint(Position p, TileKey possibility)

If an empty initializer list is passed, nothing will occur.

Parameters

p	the grid position for the initial constraint
possibilities	the tile keys which will be allowed in this grid position

5.8.4.12 setInitialConstraint() [2/2]

```
void Solver::setInitialConstraint (
            Position p,
            TileKey possibility )
```

A method to set an *initial constraint*. An initial constraint involves a grid position and a tile key. The grid square at the given position is preemptively collapsed to the given tile key, with the results being propagated, before the algorithm begins. This gives the user more say in the kinds of solutions that the algorithm will reach.

Liberal use of initial constraints may lead to frequent contradictions, so limiting their use is adviced.

Unlike the adjacency constraint API, succesive calls to this method with the same position but different tile keys does not add them all to the grid position. It is a setter, not an inserter.

Parameters

p	the grid position for the initial constraint
possibility	the sole tile key which will be allowed in this grid position

5.8.4.13 setSeed()

```
void Solver::setSeed (
            int seed )
```

Set the seed for the random number generator.

Parameters

seed	the seed

5.8.4.14 solve()

```
void Solver::solve (
            size_t N,
            Grid< TileKey > & grid )
```

Runs the wave-function collapse algorithm, solving in-place a 2-dimensional square Grid or throwing an exception if the algorithm fails. The grid modified is of dimensions NxN.

Parameters

N	The dimension of the square grid.
grid	A grid to solve

Exceptions

std::runtime_error	Indicates that the grid could not be solved.

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

- include/Solver.h
- src/Solver.cpp

5.9 Synthesizer Class Reference

The Synthesizer class.

`#include <Synthesizer.h>`

Public Member Functions

- Synthesizer ()=default

  Construct a new Synthesizer object.
- ~Synthesizer ()=default

  Destroy the Synthesizer object.
- void exportGridToFile (const SolverGrid &grid, std::string exportPath, FileType type)

  Export a completed Grid to a file of the specified format.
- std::shared_ptr< BMPImage > exportGridToImage (const SolverGrid &grid)

  Export a completed Grid to a BMPImage.
- void initRealTimeImage (unsigned int n)

  Initialize a real time image for step by step images. If there is an existing real time image, this function would clear the existing one.
- void modifyRealTimeImage (Position pos, const Tile &tile)

  Modify the real time image with one step of collapse in grid.
- std::shared_ptr< BMPImage > getRealTimeImage () const

  Get the Real Time Image object.
- void clearRealTimeImage ()

  Clear the current real time image.
- void exportRealTimeImageToFile (std::string exportPath, FileType type)

  Export the real time image to file in the specified format.

5.9.1 Detailed Description

The Synthesizer class.

This class implements the functionality of synthesizing grid into an out put image. At the moment, only png and bmp images are supported. All generated images are stored as BMPImage and transformed into specified format.

See also

Grid

BMPImage

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Synthesizer()

```
Synthesizer::Synthesizer ( )  [default]
```

Construct a new Synthesizer object.

5.9.2.2 ~Synthesizer()

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

Destroy the Synthesizer object.

5.9.3 Member Function Documentation

5.9.3.1 clearRealTimeImage()

```
void Synthesizer::clearRealTimeImage ( )
```

Clear the current real time image.

5.9.3.2 exportGridToFile()

```
void Synthesizer::exportGridToFile (
            const SolverGrid & grid,
            std::string exportPath,
            FileType type )
```

Export a completed Grid to a file of the specified format.

Parameters

grid	A completed grid of tiles.
exportPath	The path for the exported image.
type	The format for the image.

5.9.3.3 exportGridToImage()

```
std::shared_ptr< BMPImage > Synthesizer::exportGridToImage (
            const SolverGrid & grid )
```

Export a completed Grid to a BMPImage.

Parameters

grid	A completed grid of tiles

Returns

std::shared_ptr<BMPImage> Caller gets the ownership of a smart pointer to the result image.

5.9.3.4 exportRealTimeImageToFile()

```
void Synthesizer::exportRealTimeImageToFile (
            std::string exportPath,
            FileType type )
```

Export the real time image to file in the specified format.

Parameters

exportPath	The target path for the exported image.
type	The format of image.

5.9.3.5 getRealTimeImage()

```
std::shared_ptr< BMPImage > Synthesizer::getRealTimeImage ( ) const
```

Get the Real Time Image object.

Returns

std::shared_ptr<BMPImage> The caller gains ownership of the real time image.

5.9.3.6 initRealTimeImage()

```
void Synthesizer::initRealTimeImage (
            unsigned int n )
```

Initialize a real time image for step by step images. If there is an existing real time image, this function would clear the existing one.

Parameters

n	The dimension of the image.

5.9.3.7 modifyRealTimeImage()

```
void Synthesizer::modifyRealTimeImage (
            Position pos,
            const Tile & tile )
```

Modify the real time image with one step of collapse in grid.

Parameters

pos	The logical position of the target position in grid.
tile	The target tile to be filled into the position.

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

- include/Synthesizer.h
- src/Synthesizer.cpp

5.10 Tile Class Reference

The Tile class.

`#include <Tile.h>`

Public Member Functions

- Tile (const std::shared_ptr< BMPImage > img)
  Constructor for initializing with image data.
- unsigned int getSize () const
  Gets tile's size in pixels.
- std::shared_ptr< BMPImage > getImageData () const
  Gets tile's image data.
- std::vector< Position > enumeratePosition () const
  Enumerates all positions in the image.

5.10.1 Detailed Description

The Tile class.

This class is a wrapper for a BMPImage, to be used in the context of a Grid and Solver

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Tile()

```
Tile::Tile (
            const std::shared_ptr< BMPImage > img )  [inline]
```

Constructor for initializing with image data.

Exceptions

ivalid_argument	Image must be square

5.10.3 Member Function Documentation

5.10.3.1 enumeratePosition()

```
std::vector< Position > Tile::enumeratePosition ( ) const  [inline]
```

Enumerates all positions in the image.

Returns

all pixel positions

5.10.3.2 getImageData()

```
std::shared_ptr< BMPImage > Tile::getImageData ( ) const  [inline]
```

Gets tile's image data.

Returns

the image data

5.10.3.3 getSize()

```
unsigned int Tile::getSize ( ) const  [inline]
```

Gets tile's size in pixels.

Returns

the pixel size

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

- include/Tile.h

5.11 TileData Class Reference

Tile Data Class.

`#include <Extractor.h>`

Public Attributes

- int **id**
- unsigned int **width**
- unsigned int **height**
- std::vector< Pixel > **pixels**
- std::vector< Pixel > **north**
- std::vector< Pixel > **south**
- std::vector< Pixel > **east**
- std::vector< Pixel > **west**
- std::map< int, std::set< int > > **northConstraints**
- std::map< int, std::set< int > > **southConstraints**
- std::map< int, std::set< int > > **westConstraints**
- std::map< int, std::set< int > > **eastConstraints**

5.11.1 Detailed Description

Tile Data Class.

Author

Ricardo Figueroa

Contains all the required TileData retrieved through the extraction. Most of the names are self explanitory. Id, width, height, pixels... The different sides (north, south, east, west) however refer to the list of pixels that encompase those sides for the tile. The side constraints however, contain a set of ids mapped to the id of the tile.

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

- include/Extractor.h

Chapter 6

File Documentation

6.1 BMPImage.h

```
1 #pragma once
2
3 #include <vector>
4 #include <string>
5
6 #include <Position.h>
7 #include <FileType.h>
8
9 #include <Pixel.h>
10
11 constexpr int defaultHorizontalDPI = 96;
12 constexpr int defaultVerticalDPI = 96;
13 constexpr int defaultBitDepth = 32;
14
26 class BMPImage {
27
28 public:
29
34     BMPImage();
35
42     BMPImage(unsigned int initWidth, unsigned int initHeight);
43
49     BMPImage(std::string filename);
50
55     ~BMPImage() = default;
56
57
58     /* Size */
59
65     unsigned int getWidth() const noexcept;
66
72     unsigned int getHeight() const noexcept;
73
81     void setSize(unsigned int NewWidth , unsigned int NewHeight);
82
83     /* Pixel */
84
91     Pixel getPixel(Position pos) const;
92
100     void setPixel(Position pos, Pixel newPixel);
101
109     void exportToFile(std::string filename, FileType type) const;
110
111 private:
112
113     /* PIXELS */
114     std::vector<std::vector<Pixel>> pixels;
115
116     /* METADATA */
117     int bitDepth;
118     int verticalDPI, horizontalDPI;
119
120     /* HELPERS */
121     inline bool checkPosition(Position pos) const;
122
123 };
124
134 inline bool operator==(const BMPImage& lhs, const BMPImage& rhs)
```

```
135 {
136     if (lhs.getWidth() != rhs.getWidth() || lhs.getHeight() != rhs.getHeight()) {
137         return false;
138     }
139 
140     for (unsigned int x = 0; x < lhs.getWidth(); x++)
141         for (unsigned int y = 0; y < lhs.getHeight(); y++) {
142             if (lhs.getPixel({x, y}) != rhs.getPixel({x, y})) {
143                 return false;
144             }
145         }
146     return true;
147 }
148 
```

6.2 Direction.h

```
1 #pragma once
2 
7 enum Direction {
8     UP,
9     DOWN,
10     LEFT,
11     RIGHT
12 };
```

6.3 Extractor.h

```
1 #pragma once
2 
3 #include <Pixel.h>
4 
5 #include <vector>
6 #include <string>
7 #include <map>
8 #include <set>
9 
10 
11 enum Side { north, south, east, west };
12 
27 class TileData {
28     public:
29         int id;
30         unsigned int width, height;
31         std::vector<Pixel> pixels;
32 
33         std::vector<Pixel> north;
34         std::vector<Pixel> south;
35         std::vector<Pixel> east;
36         std::vector<Pixel> west;
37 
38         std::map<int, std::set<int» northConstraints;
39         std::map<int, std::set<int» southConstraints;
40         std::map<int, std::set<int» westConstraints;
41         std::map<int, std::set<int» eastConstraints;
42 };
43 
44 
89 class Extractor {
90     public:
91 
102         void extractPNG(unsigned int& width, unsigned int& height, const std::string& filename,
      std::vector<Pixel>& pixels);
103 
114         void extractBMP(unsigned int& width, unsigned int& height, const std::string& filename,
      std::vector<Pixel>& pixels);
115 
128         int extractTileset(unsigned int width, unsigned int height, unsigned int horizontal, unsigned
      int vertical, const std::vector<Pixel>& pixels, std::vector<TileData>& tiles);
129 
137         void encodeTilesetPNG(std::vector<TileData>& tiles, std::string assetName);
138 
139 
150         static void encodePNG(unsigned int width, unsigned int height, const std::string& filename,
      const std::vector<Pixel>& pixels);
151 
159         int setConstraints(std::vector<TileData>& tileList);
160 
167         void setComparisonMetric(double m);
```

```
168
176         void setDumpDirectoryRelative(std::string dir);
177
185         void setDumpDirectoryAbsolute(std::string dir);
186
194         void setAssetDirectoryRelative(std::string dir);
195
203         void setAssetDirectoryAbsolute(std::string dir);
204
209         void printDirectories();
210
211
218         void printConstraintsV(std::vector<TileData> tiles);
219
226         void printConstraints(TileData tile);
227
232         Extractor(){
233             idCount = 1;
234             comparisonMetric = .8;
235         }
236
244         Extractor(double metric){
245             idCount = 1;
246             if(metric > 1){
247                 metric = 1;
248             }
249             if(metric < 0){
250                 metric = 0;
251             }
252
253             comparisonMetric = metric;
254         }
255
256     private:
257
258             int idCount;
259         double comparisonMetric;
260         std::string aDir;
261         std::string dDir;
262
263
275         int tileFormation(const std::vector<Pixel>& pixels, TileData& tile, unsigned int width, unsigned
      int height, int id);
276
285         int tileCompare(TileData& tile1, TileData& tile2);
286
295         bool sideCompare(std::vector<Pixel> side1, std::vector<Pixel> side2, unsigned int length);
296
297 }
298 ;
```

6.4 FileType.h

```
1 #pragma once
2
3 enum FileType {
4     bmp,
5     png
6 };
```

6.5 Grid.h

```
1 #pragma once
2
3 #include <Tile.h>
4 #include <Position.h>
5
6 #include <map>
7 #include <vector>
8 #include <limits.h>
9
10 template <typename TileKey>
11 class Grid {
12
13 public:
14     Grid() { dimension = 0; }
15
16     ~Grid() = default;
17
```

```
18      Grid(size_t dimension, std::map<TileKey, std::shared_ptr<Tile» map)
19      {
20          setDimension(dimension);
21          setTileMap(map);
22      }
23
24      Grid(size_t dimension)
25      {
26          setDimension(dimension);
27      }
28
29      size_t getDimension() const
30      {
31          return dimension;
32      }
33
34      void setDimension(size_t newDimension)
35      {
36          dimension = newDimension;
37          tileKeyGrid.resize(newDimension, std::vector<TileKey>());
38          for (auto& row : tileKeyGrid)
39              row.resize(newDimension, TileKey(-1)); // suppose -1 == unassigned
40      }
41
42      TileKey getKey(Position p)
43      {
44          return tileKeyGrid[p.x][p.y];
45      }
46
47      void setKey(Position p, TileKey key)
48      {
49          if (!checkPosition(p))
50              throw std::out_of_range("Position out of range. ");
51
52          tileKeyGrid[p.y][p.x] = key;
53      }
54
55      std::shared_ptr<Tile> getTile(Position pos) const
56      {
57          if (!checkPosition(pos))
58              throw std::out_of_range("Position out of range. ");
59
60          auto tileKey = tileKeyGrid[pos.y][pos.x];
61          auto tile = tileMap.at(tileKey);
62
63          return tile;
64      }
65
66      void setTile(Position pos, TileKey tileKey)
67      {
68          if (!tileMap.contains(tileKey))
69              throw std::out_of_range("TileKey does not exist. ");
70
71          if (!checkPosition(pos))
72              throw std::out_of_range("Position out of range. ");
73
74          tileKeyGrid[pos.y][pos.x] = tileKey;
75      }
76
77      void setTileMap(std::map<TileKey, std::shared_ptr<Tile» newMap) {
78          for(const auto & e : newMap)
79          {
80              // If the key exists, change only the value, add the {key, value} otherwise
81              tileMap[e.first] = e.second;
82          }
83      }
84
85      Position translatePixelPosition(Position pos) const
86      {
87          if (!checkPosition(pos))
88              throw std::out_of_range("Position out of range. ");
89
90          if (getTile(pos)->getSize() > INT_MAX)
91              throw std::runtime_error("Tile size too big. ");
92
93          auto tileSize = getTile(pos)->getSize();
94
95          return { pos.x * tileSize, pos.y * tileSize} ;
96      }
97
98      std::vector<Position> enumeratePosition() const
99      {
100          std::vector<Position> res;
101
102         for (size_t i = 0; i < dimension; i++)
103             for (size_t j = 0; j < dimension; j++) {
104                 Position pos = {j, i};
```

```
105                 res.push_back(pos);
106             }
107 
108         return res;
109     }
110 
111 private:
112 
113     size_t dimension;
114 
115     std::map<TileKey, std::shared_ptr<Tile» tileMap;
116 
117     std::vector<std::vector<TileKey» tileKeyGrid;
118 
119     constexpr bool checkPosition(Position pos) const
120     {
121         size_t i = pos.x;
122         size_t j = pos.y;
123         return (i < dimension && i >= 0 && j < dimension && j >= 0);
124     }
125 };
126 
```

6.6 Pixel.h

```
1 #pragma once
2 
3 #include <sstream>
4 
9 typedef unsigned char BYTE;
10 
17 template<typename T>
18 concept PixelType = requires
19 {
20     T::Red;
21     T::Green;
22     T::Blue;
23     T::Alpha;
24 };
25 
30 typedef struct Pixel {
31     int count;
32 
37     BYTE Blue;
38 
43     BYTE Green;
44 
49     BYTE Red;
50 
55     BYTE Alpha;
56 
66     template<PixelType T>
67     Pixel& operator=(const T& rhs) {
68         Alpha = rhs.Alpha;
69         Blue = rhs.Blue;
70         Green = rhs.Green;
71         Red = rhs.Red;
72         count = 0;
73         return *this;
74     }
75 
76 } Pixel;
77 
87 inline bool operator==(const Pixel& lhs, const Pixel& rhs) {
88     return lhs.Pixel::Blue == rhs.Pixel::Blue && lhs.Pixel::Green == rhs.Pixel::Green && lhs.Red ==
       rhs.Red && lhs.Alpha == rhs.Alpha;
89 }
90 
99 std::ostream& operator«(std::ostream &os, const Pixel &p);
100 
101 /* DEFAULT CONSTANTS */
102 
103 constexpr Pixel defaultPixel = {
104     0,
105     255,
106     255,
107     255,
108     0
109 };
```

6.7 Position.h

```
1 #pragma once
2
3 #include <Direction.h>
4
5 #include <cstddef>
6
11 struct Position {
12     size_t x;
13     size_t y;
14
22     Direction getDirection(Position other) {
23       if (other.x > x)
24         return Direction::RIGHT;
25       else if (other.x < x)
26         return Direction::LEFT;
27       else if (other.y > y)
28         return Direction::UP;
29       else
30         return Direction::DOWN;
31     }
32 };
33
34 inline bool operator <(const Position& lhs, const Position& rhs){
35     return lhs.x >= rhs.x ? lhs.y < rhs.y : true;
36 }
37
38 inline bool operator==(const Position& lhs, const Position& rhs) {
39     return lhs.x == rhs.x && lhs.y == rhs.y;
40 }
```

6.8 Rotation.h

```
1 #pragma once
2
3 enum Rotation {
4     D0,
5     D90,
6     D180,
7     D270
8 };
```

6.9 Solver.h

```
1 #pragma once
2
3 #include <Tile.h>
4 #include <Grid.h>
5 #include <Position.h>
6
7 #include <memory>
8 #include <map>
9 #include <set>
10 #include <vector>
11 #include <list>
12 #include <functional>
13 #include <unordered_set>
14 #include <unordered_map>
15 #include <stack>
16 #include <utility>
17 #include <tuple>
18 #include <optional>
19 #include <iterator>
20 #include <algorithm>
21 #include <cstdlib>
22 #include <stdexcept>
23
24
25 template <>
26 struct std::hash<Position> {
27   std::size_t operator()(const Position& k) const {
28     return (k.y « 16) ^ k.x;
29   }
30 };
31 template <>
32 struct std::hash<std::pair<size_t, Direction» {
33   std::size_t operator()(const std::pair<size_t, Direction>& k) const {
34     return k.first ^ (size_t)k.second;
```

```
35   }
36 };
37
73 class Solver {
74 public:
75
79   typedef size_t TileKey;
80
81   typedef std::function<std::vector<TileKey>::const_iterator(const std::vector<TileKey>&)>
       CollapseBehavior;
82
83   typedef std::function<void(const TileKey&, Position)> CollapseCallback;
84   typedef std::function<void(const std::vector<TileKey>&, Position)> PropagateCallback;
85
90   typedef typename std::list<CollapseCallback>::iterator CollapseCallbackCookie;
91
96   typedef typename std::list<PropagateCallback>::iterator PropagateCallbackCookie;
97
103   template<typename T>
104   Solver(const std::vector<T>& tiles, int seed=0): seed(seed) {
105     for (TileKey k = 0; k < tiles.size(); k++) {
106       this->tiles.push_back(k);
107     }
108   }
109
115   template<typename T>
116   Solver(const std::map<TileKey, T>& tiles, int seed=0): seed(seed) {
117     for (auto [k, _] : tiles) {
118       this->tiles.push_back(k);
119     }
120   }
121
122   /*
123    SOLVER API
124   */
125
130   void setSeed(int seed);
131
136   int getSeed();
137
145   void solve(size_t N, Grid<TileKey>& grid);
146
147   /*
148    CONSTRAINT INTERFACE/API
149    Set the solver constraints using this specified interface
150   */
151
168   void addAdjacencyConstraint(TileKey t, Direction d, TileKey neighbor);
169
180   void removeAdjacencyConstraint(TileKey t, Direction d, TileKey neighbor);
181
193   void addAdjacencyConstraint(TileKey t, Direction d, std::initializer_list<TileKey> neighbors);
194
206   void removeAdjacencyConstraint(TileKey t, Direction d, std::initializer_list<TileKey> neighbors);
207
222   void setInitialConstraint(Position p, TileKey possibility);
223
232   void setInitialConstraint(Position p, std::initializer_list<TileKey> possibilities);
233
234   /*
235    CALLBACK INTERFACE/API
236   */
237
249   CollapseCallbackCookie registerOnCollapse(CollapseCallback callback);
250
258   void deregisterOnCollapse(CollapseCallbackCookie cookie);
259
271   PropagateCallbackCookie registerOnPropagate(PropagateCallback callback);
272
280   void deregisterOnCollapse(PropagateCallbackCookie cookie);
281
282   /*
283    COLLAPSE INTERFACE
284   */
285
293   void setCollapseBehaviour(std::optional<CollapseBehavior> b);
294
295
296 private:
297   typedef std::pair<TileKey, Direction> Side;
298
299   /* INITIAL */
300   int seed;
301   std::vector<TileKey> tiles;
302
303   /* CONSTRAINTS */
304   std::unordered_map<Side, std::unordered_set<TileKey» adjacency_constraints;
```

```
305   std::unordered_map<Position, std::unordered_set<TileKey» initial_constraints;
306
307   /* BEHAVIOURS */
308   CollapseBehavior collapse_behavior = nullptr;
309
310   /* CALLBACKS */
311   std::list<CollapseCallback> collapse_callbacks;
312   std::list<PropagateCallback> propagate_callbacks;
313
314   /* ALGORITHM */
315   std::unordered_map<Position, std::vector<TileKey» grid;
316   size_t N;
317
318   void initializeGrid(size_t N);
319   void processInitialConstraints() ;
320   bool isCollapsed();
321   bool isContradiction();
322   void iterate();
323   Position getMinEntropyCoordinates();
324   void collapseAt(Position p);
325   void propagate(Position p);
326   bool propagateAt(Position current, Position neighbor); //returns true if neighbor's possible tiles
       decrease
327   std::vector<Position> getNeighbors(Position p);
328   std::vector<TileKey>::const_iterator collapseRandom(const std::vector<TileKey>& tiles);
329   std::unordered_set<TileKey> getAdjacencies(TileKey k, Direction d);
330   std::vector<TileKey> getPossibleTiles(Position p);
331
332 };
333
339 typedef Grid<Solver::TileKey> SolverGrid;
340
```

6.10 Synthesizer.h

```
1 #pragma once
2
3 #include <Solver.h>
4 #include <FileType.h>
5 #include <memory>
6
7
19 class Synthesizer {
20
21 public:
22
27     Synthesizer() = default;
28
33     ~Synthesizer() = default;
34
41     void exportGridToFile(const SolverGrid& grid, std::string exportPath, FileType type);
42
49     std::shared_ptr<BMPImage> exportGridToImage(const SolverGrid& grid);
50
57     void initRealTimeImage(unsigned int n);
58
65     void modifyRealTimeImage(Position pos, const Tile& tile);
66
72     std::shared_ptr<BMPImage> getRealTimeImage() const;
73
78     void clearRealTimeImage();
79
86     void exportRealTimeImageToFile(std::string exportPath, FileType type);
87
88
89 private:
90
91     /* IMAGE STATE */
92     std::shared_ptr<BMPImage> realTimeImage;
93
94     /* HELPER */
95     void copyTileToGrid(Position pos, const Tile& tile, BMPImage* gridImage);
96
97 };
```

6.11 Tile.h

```
1 #pragma once
2
```

```
3 #include <Direction.h>
4 #include <Position.h>
5 #include <BMPImage.h>
6
7 #include <string>
8 #include <memory>
9
10
18 class Tile {
19
20 public:
21
22     Tile() = default;
23     ~Tile() = default;
24
29     Tile(const std::shared_ptr<BMPImage> img)
30     {
31         if (img->getHeight() != img->getWidth())
32             throw std::invalid_argument("Image must be square. ");
33         size = img->getHeight();
34         image = img;
35     }
36
41     unsigned int getSize() const
42     {
43         return size;
44     }
45
51     std::shared_ptr<BMPImage> getImageData() const
52     {
53         return image;
54     }
55
61     std::vector<Position> enumeratePosition() const
62     {
63         std::vector<Position> res;
64
65         for (unsigned int i = 0; i < size; i++)
66             for (unsigned int j = 0; j < size; j++) {
67                 Position pos = {j, i};
68                 res.push_back(pos);
69             }
70
71         return res;
72     }
73
74 private:
75
76     std::string filePath;
77     // TODO: add other variables
78
79     unsigned int size;
80     std::shared_ptr<BMPImage> image;
81 };
```