Spotlight

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces 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

Namespace Documentation

4.1 PathUtils Namespace Reference

Contains helper functions for resolving application file paths.

Functions

- std::filesystem::path getExecutableDir ()

  Retrieves the directory where the executable is located.
- std::filesystem::path getConfigPath (const std::string &relativeFile="")

  Constructs a path within the "config" directory.
- std::filesystem::path getAssetPath (const std::string &relativeFile="")

  Constructs a path within the "assets" directory.

4.1.1 Detailed Description

Contains helper functions for resolving application file paths.

4.1.2 Function Documentation

4.1.2.1 getAssetPath()

```
std::filesystem::path PathUtils::getAssetPath (
            const std::string & relativeFile = "")
```

Constructs a path within the "assets" directory.

Parameters

relativeFile	Optional relative file or subpath within "assets".

Returns

The full asset path.

Here is the call graph for this function: Here is the caller graph for this function:

4.1.2.2 getConfigPath()

```
std::filesystem::path PathUtils::getConfigPath (
            const std::string & relativeFile = "")
```

Constructs a path within the "config" directory.

Parameters

relativeFile	Optional relative file or subpath within "config".

Returns

The full configuration path.

Here is the call graph for this function:

4.1.2.3 getExecutableDir()

```
std::filesystem::path PathUtils::getExecutableDir ()
```

Retrieves the directory where the executable is located.

Returns

The path to the executable's directory.

Here is the caller graph for this function:

Chapter 5

Class Documentation

5.1 Board Class Reference

Represents a hexagonal game board composed of cube-coordinate tiles.

`#include <Board.hpp>`

Public Member Functions

- Board (int radius)

  Constructs a new hexagonal board with the given radius.
- Tile * getTile (const CubeCoord &coord)

  Retrieves a pointer to the tile at the specified cube coordinate.
- std::vector< CubeCoord > getNeighbors (const CubeCoord &coord) const

  Returns all valid neighboring coordinates of the specified tile.
- void setTileOwner (int x, int y, int z, Company *company)

  Assigns a company as the owner of a tile at the given coordinates.
- void setTileColor (int x, int y, int z, const std::string &color)

  Assigns a color to the tile at the specified coordinates.
- void printBoard () const

  Prints a visual or textual representation of the board.

Public Attributes

- std::unordered_map< CubeCoord, Tile, CubeCoordHash > tiles

  Map of cube coordinates to their corresponding Tile instances.

5.1.1 Detailed Description

Represents a hexagonal game board composed of cube-coordinate tiles.

The Board is implemented as an unordered map keyed by CubeCoord, enabling constant-time lookup of tiles. Each tile may store its owner (Company) and visual color data. The board supports neighbor detection and manipulation of tile attributes.

The design is suitable for strategy or simulation games utilizing hex grids.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Board()

```
Board::Board (
            int radius)
```

Constructs a new hexagonal board with the given radius.

Initializes a new Board and immediately generates its tiles.

Parameters

in	*radius*	The radius of the hex grid (number of tiles from center to edge).

Precondition

radius $>$ 0

Postcondition

Initializes a complete set of valid cube coordinates within the given radius.

5.1.3 Member Function Documentation

5.1.3.1 getNeighbors()

```
std::vector< CubeCoord > Board::getNeighbors (
            const CubeCoord & coord) const
```

Returns all valid neighboring coordinates of the specified tile.

Parameters

in	*coord*	The cube coordinate for which to find adjacent tiles.

Returns

Vector of CubeCoord objects representing neighboring tiles.

Postcondition

The returned vector contains only coordinates that exist within the board radius.

See also

CubeCoord

Iterates through the six cube-coordinate directions to find all existing adjacent tiles. Excludes invalid or out-of-bounds tiles.

5.1.3.2 getTile()

```
Tile * Board::getTile (
            const CubeCoord & coord)
```

Retrieves a pointer to the tile at the specified cube coordinate.

Returns a pointer to the Tile at the given cube coordinate, or nullptr if missing.

Parameters

in	*coord*	The cube coordinate to locate.

Returns

Pointer to the Tile if it exists, or nullptr if out of bounds.

Precondition

The coordinate must satisfy $x + y + z = 0$ for valid hex geometry.

Postcondition

Does not modify board state.

Here is the caller graph for this function:

5.1.3.3 printBoard()

```
void Board::printBoard () const
```

Prints a visual or textual representation of the board.

Implementation-defined visualization; typically for debugging or display. May include coordinates, colors, or ownership data.

Postcondition

Does not modify the board state.

5.1.3.4 setTileColor()

```
void Board::setTileColor (
            int x,
            int y,
            int z,
            const std::string & color)
```

Assigns a color to the tile at the specified coordinates.

Parameters

in	*x*	Cube X-coordinate.
in	*y*	Cube Y-coordinate.
in	*z*	Cube Z-coordinate.
in	*color*	String representation of the desired color.

Precondition

The tile at (x, y, z) must exist and the color string must be valid.

Postcondition

The tile's color is updated.

Warning

Prints an error if the tile does not exist or color is invalid.

See also

Colors::isValid()

Validates the target coordinate and color before assignment. Logs errors if either is invalid but continues execution safely. Here is the call graph for this function:

5.1.3.5 setTileOwner()

```
void Board::setTileOwner (
            int x,
            int y,
            int z,
            Company * company)
```

Assigns a company as the owner of a tile at the given coordinates.

Parameters

in	*x*	Cube X-coordinate.
in	*y*	Cube Y-coordinate.
in	*z*	Cube Z-coordinate.
in,out	*company*	Pointer to the Company to assign as owner.

Precondition

The specified coordinate must exist on the board.

Postcondition

The tile's ownership is updated.

Warning

Does nothing if the coordinate is invalid.

Updates the ownership of a specific tile if it exists. Useful for territory claiming or resource control systems. Here is the call graph for this function:

5.1.4 Member Data Documentation

5.1.4.1 tiles

```
std::unordered_map<CubeCoord, Tile, CubeCoordHash> Board::tiles
```

Map of cube coordinates to their corresponding Tile instances.

The unordered_map allows efficient random access and modification of tiles.

See also

CubeCoord, Tile

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

- include/Board.hpp
- src/Board.cpp

5.2 Card Class Reference

Represents a playable card with triggers and associated actions.

```
#include <Card.hpp>
```

Public Member Functions

- **Card** ()=default

  Default constructor. Initializes an empty card.
- Card (const nlohmann::json &data)

  Constructs a card from parsed JSON data.
- void executeTrigger (const std::string &trigger, class Player &player) const

  Executes all actions tied to a given trigger.

Public Attributes

- std::string name
- std::string description
- std::unordered_map< std::string, std::vector< nlohmann::json > > triggers

  Maps trigger event names to a list of action definitions.

5.2.1 Detailed Description

Represents a playable card with triggers and associated actions.

Each card contains a name, description, and a set of triggers that define when certain actions occur (e.g., "onPlay", "onStartOfDay"). Actions are stored as JSON objects that can be executed dynamically.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Card()

```
Card::Card (
            const nlohmann::json & data)
```

Constructs a card from parsed JSON data.

Parameters

data	JSON object containing card properties and triggers.

5.2.3 Member Function Documentation

5.2.3.1 executeTrigger()

```
void Card::executeTrigger (
            const std::string & trigger,
            class Player & player) const
```

Executes all actions tied to a given trigger.

Parameters

trigger	The trigger event name (e.g. "onPlay").
player	The player affected by this card's actions.

Here is the call graph for this function:

5.2.4 Member Data Documentation

5.2.4.1 description

```
std::string Card::description
```

Text description or effect summary.

5.2.4.2 name

```
std::string Card::name
```

The display name of the card.

5.2.4.3 triggers

```
std::unordered_map<std::string, std::vector<nlohmann::json> > Card::triggers
```

Maps trigger event names to a list of action definitions.

Example format:

```
{
  "onPlay": [
    { "action": "addResource", "type": "funds", "amount": 2 }
  ]
}
```

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

- include/Card.hpp
- src/Card.cpp

5.3 Colors Class Reference

Centralized static utility for color name validation and mapping.

`#include <Colors.hpp>`

Static Public Member Functions

- static bool isValid (const std::string &color)
  Checks whether a color name is valid.
- static const sf::Color & getSfmlColor (const std::string &color)
  Retrieves the SFML color object associated with a color name.

Static Public Attributes

- static const std::map< std::string, std::string > colorCodes
  Maps color names to their ANSI terminal escape codes.
- static const std::map< std::string, sf::Color > sfmlColors
  Maps color names to their SFML color equivalents.
- static const std::vector< std::string > all
  List of all valid color names.

5.3.1 Detailed Description

Centralized static utility for color name validation and mapping.

Provides a list of all valid color names, their ANSI escape codes for console output, and their corresponding SFML color values for rendering.

5.3.2 Member Function Documentation

5.3.2.1 getSfmlColor()

```
const sf::Color & Colors::getSfmlColor (
            const std::string & color)  [static]
```

Retrieves the SFML color object associated with a color name.

Parameters

color	The color name to retrieve.

Returns

A reference to the matching sf::Color, or the "Neutral" color if not found.

5.3.2.2 isValid()

```
bool Colors::isValid (
            const std::string & color)  [static]
```

Checks whether a color name is valid.

Parameters

color	The color name to check.

Returns

True if the color exists in all, false otherwise.

Here is the caller graph for this function:

5.3.3 Member Data Documentation

5.3.3.1 all

```
const std::vector< std::string > Colors::all  [static]
```

Initial value:

```
= {
    "Red", "Yellow", "Blue", "Green", "Purple", "White", "Gray", "Neutral"
}
```

List of all valid color names.

5.3.3.2 colorCodes

```
const std::map< std::string, std::string > Colors::colorCodes  [static]
```

Initial value:

```
= {
    {"Red",     "\033[41m  \033[0m"},
    {"Yellow",  "\033[43m  \033[0m"},
    {"Blue",    "\033[44m  \033[0m"},
    {"Green",   "\033[42m  \033[0m"},
    {"Purple",  "\033[45m  \033[0m"},
    {"White",   "\033[47m  \033[0m"},
    {"Gray",    "\033[100m  \033[0m"},
    {"Neutral", "\033[0m..."}
}
```

Maps color names to their ANSI terminal escape codes.

Example: "Red" → "\033[41m \033[0m"

5.3.3.3 sfmlColors

```
const std::map< std::string, sf::Color > Colors::sfmlColors  [static]
```

Initial value:

```
= {
    {"Red",     sf::Color(0xD9, 0x7B, 0x66)},
    {"Yellow",  sf::Color(0xE3, 0xC5, 0x67)},
    {"Blue",    sf::Color(0x6C, 0x8E, 0xBF)},
    {"Green",   sf::Color(0x7C, 0xA9, 0x82)},
    {"Purple",  sf::Color(0xA8, 0x8E, 0xC6)},
    {"White",   sf::Color(0xF2, 0xE9, 0xE4)},
    {"Gray",    sf::Color(0xB0, 0xA8, 0xB9)},
    {"Neutral", sf::Color(0x4B, 0x4A, 0x54)}
}
```

Maps color names to their SFML color equivalents.

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

- include/Colors.hpp
- src/Colors.cpp

5.4 CommandConsole Class Reference

In-game text console handling user command input, output history, and paging.

`#include <CommandConsole.hpp>`

Public Member Functions

- CommandConsole (Board &b, sf::Font &f, sf::Vector2f pos)
  Constructs a command console linked to a board.
- void draw (sf::RenderWindow &window)
  Renders the console contents to the window.
- void handleEvent (const sf::Event &e)
  Handles keyboard input events to build commands and navigate history.
- void print (const std::string &line)
  Prints a single line of text to the console output.
- void printPaged (const std::vector< std::string > &lines)
  Displays multiple lines of text using a paginated system.
- void **showNextPage** ()
  Advances to and displays the next page of paginated text, if available.
- std::string nextCommand ()
  Retrieves the next user-entered command.
- void **clear** ()
  Clears all console output lines.
- bool hasCommand () const
  Returns whether there are commands pending to process.

Public Attributes

- bool awaitingNextPage = false

5.4.1 Detailed Description

In-game text console handling user command input, output history, and paging.

The CommandConsole manages user input through keyboard events, maintains command history, supports paginated output, and displays text using SFML rendering.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 CommandConsole()

```
CommandConsole::CommandConsole (
            Board & b,
            sf::Font & f,
            sf::Vector2f pos)
```

Constructs a command console linked to a board.

Parameters

b	Reference to the Board this console can interact with.
f	Reference to the SFML font used for text rendering.
pos	Screen position where the console text begins.

5.4.3 Member Function Documentation

5.4.3.1 draw()

```
void CommandConsole::draw (
            sf::RenderWindow & window)
```

Renders the console contents to the window.

Parameters

window	Reference to the SFML render window.

Here is the caller graph for this function:

5.4.3.2 handleEvent()

```
void CommandConsole::handleEvent (
            const sf::Event & e)
```

Handles keyboard input events to build commands and navigate history.

Parameters

e	The SFML event to process.

Here is the caller graph for this function:

5.4.3.3 hasCommand()

```
bool CommandConsole::hasCommand () const  [inline]
```

Returns whether there are commands pending to process.

Returns

True if one or more commands are waiting in the queue.

5.4.3.4 nextCommand()

```
std::string CommandConsole::nextCommand ()
```

Retrieves the next user-entered command.

Returns

The command string at the front of the pending command queue.

5.4.3.5 print()

```
void CommandConsole::print (
            const std::string & line)
```

Prints a single line of text to the console output.

Parameters

line	Text line to display.

Here is the caller graph for this function:

5.4.3.6 printPaged()

```
void CommandConsole::printPaged (
            const std::vector< std::string > & lines)
```

Displays multiple lines of text using a paginated system.

Parameters

lines	A vector of lines to show, split into multiple pages if necessary.

Here is the call graph for this function:

5.4.4 Member Data Documentation

5.4.4.1 awaitingNextPage

```
bool CommandConsole::awaitingNextPage = false
```

True if more paginated output remains.

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

- include/CommandConsole.hpp
- src/CommandConsole.cpp

5.5 Company Class Reference

Represents a company, corporation, or in-game faction.

`#include <Company.hpp>`

Public Member Functions

- Company (const std::string &name, const std::string &symbol)
  Constructs a new Company with the specified name and symbol.
- const std::string & getName () const
  Retrieves the company's full name.
- const std::string & getSymbol () const
  Retrieves the company's identifying symbol.
- void setName (const std::string &newName)
  Updates the company's name.
- void setSymbol (const std::string &newSymbol)
  Updates the company's identifying symbol.

5.5.1 Detailed Description

Represents a company, corporation, or in-game faction.

The Company class stores immutable identifiers that describe an entity's identity and branding in the simulation or game world. Its instances can be linked to tiles, assets, or scores via references or pointers.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Company()

```
Company::Company (
            const std::string & name,
            const std::string & symbol)
```

Constructs a new Company with the specified name and symbol.

Parameters

in	*name*	The full display name of the company.
in	*symbol*	A short identifying symbol or abbreviation.

Precondition

Both name and symbol should be non-empty strings.

Postcondition

Initializes the company with the provided identifiers.

Constructs the Company with a designated name and symbol. Typically called during setup or game initialization.

5.5.3 Member Function Documentation

5.5.3.1 getName()

```
const std::string & Company::getName () const
```

Retrieves the company's full name.

Returns a constant reference to the company's name.

Returns

A constant reference to the company's name.

Postcondition

Does not modify the object.

Here is the caller graph for this function:

5.5.3.2 getSymbol()

```
const std::string & Company::getSymbol () const
```

Retrieves the company's identifying symbol.

Returns a constant reference to the company's identifying symbol.

Returns

A constant reference to the company's symbol string.

Postcondition

Does not modify the object.

5.5.3.3 setName()

```
void Company::setName (
            const std::string & newName)
```

Updates the company's name.

Parameters

in	*newName*	The new name to assign.

Postcondition

The internal name string is replaced with newName.

Warning

Should be used cautiously if company instances are referenced elsewhere.

Allows dynamic renaming of a company, useful in sandbox or modifiable environments.

5.5.3.4 setSymbol()

```
void Company::setSymbol (
            const std::string & newSymbol)
```

Updates the company's identifying symbol.

Parameters

in	*newSymbol*	The new symbol to assign.

Postcondition

The internal symbol string is replaced with newSymbol.

Warning

Use carefully if the symbol is used as a unique key elsewhere.

Updates the company symbol. Intended for cosmetic or branding adjustments.

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

- include/Company.hpp
- src/Company.cpp

5.6 CubeCoord Struct Reference

Represents a position in a cube coordinate system.

`#include <CubeCoord.hpp>`

Public Member Functions

- CubeCoord (int x_, int y_, int z_)

  Constructs a CubeCoord with the specified coordinates.
- bool operator== (const CubeCoord &other) const

  Compares two CubeCoords for equality.
- CubeCoord operator+ (const CubeCoord &other) const

  Adds two CubeCoords component-wise.

Public Attributes

- int **x**

  X-axis coordinate.
- int **y**

  Y-axis coordinate.
- int **z**

  Z-axis coordinate.

5.6.1 Detailed Description

Represents a position in a cube coordinate system.

Each cube coordinate satisfies x + y + z = 0 in a hex grid context. Provides equality and addition operators for coordinate manipulation.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 CubeCoord()

```
CubeCoord::CubeCoord (
            int x_,
            int y_,
            int z_)  [inline]
```

Constructs a CubeCoord with the specified coordinates.

Parameters

*x_*	The x coordinate.
*y_*	The y coordinate.
*z_*	The z coordinate.

Here is the caller graph for this function:

5.6.3 Member Function Documentation

5.6.3.1 operator+()

```
CubeCoord CubeCoord::operator+ (
            const CubeCoord & other) const
```

Adds two CubeCoords component-wise.

Parameters

other	The coordinate to add.

Returns

The resulting CubeCoord.

Parameters

other	The coordinate to add.

Returns

New CubeCoord representing the sum of the two coordinates.

< Component-wise addition.Here is the call graph for this function:

5.6.3.2 operator==()

```
bool CubeCoord::operator== (
            const CubeCoord & other) const
```

Compares two CubeCoords for equality.

Parameters

other	The coordinate to compare against.

Returns

True if both coordinates are equal.

Parameters

other	The coordinate to compare against.

Returns

True if all components (x, y, z) are equal.

< Return true only if all coordinates match.Here is the call graph for this function:

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

- include/CubeCoord.hpp
- src/CubeCoord.cpp

5.7 CubeCoordHash Struct Reference

Hash functor for CubeCoord, allowing use in unordered containers.

`#include <CubeCoord.hpp>`

Public Member Functions

- std::size_t operator() (const CubeCoord &c) const noexcept
  Generates a hash for a CubeCoord.

5.7.1 Detailed Description

Hash functor for CubeCoord, allowing use in unordered containers.

5.7.2 Member Function Documentation

5.7.2.1 operator()()

```
std::size_t CubeCoordHash::operator() (
            const CubeCoord & c) const  [noexcept]
```

Generates a hash for a CubeCoord.

Generates a hash for a CubeCoord to be used in unordered containers.

Parameters

c	The coordinate to hash.

Returns

The hash value.

Parameters

c	The coordinate to hash.

Returns

Combined hash value for x, y, z.

Combines individual hashes using bit shifting to reduce collisions. < Hash x component.

< Hash y component.

< Hash z component.

< Combine hashes with bit shifts.

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

- include/CubeCoord.hpp
- src/CubeCoord.cpp

5.8 Deck Class Reference

Represents a collection of cards and provides utility operations for card management.

`#include <Deck.hpp>`

Public Member Functions

- Deck (const std::string &deckName="")
  Constructs a deck with an optional name.
- void loadFromJsonFile (const std::string &filename)
  Loads cards into the deck from a JSON file.
- void addCard (const Card &card)
  Adds a card to the deck.
- Card drawCard ()
  Draws the topmost card from the deck.
- void moveCardTo (const Card &card, Deck &deck)
  Transfers a card to another deck.
- void shuffle ()
  Randomly shuffles the order of cards in the deck.
- size_t size () const
  Gets the number of cards currently in the deck.
- bool empty () const
  Checks whether the deck is empty.

Public Attributes

- std::string **name**

  The deck's name, used for identification or debugging.
- std::vector< Card > **cards**

  The internal storage of cards contained within the deck.

5.8.1 Detailed Description

Represents a collection of cards and provides utility operations for card management.

The Deck class encapsulates a vector of Card objects along with a name identifier. It supports loading cards from external JSON data, shuffling the deck, drawing cards, and transferring them to other decks. The design is extensible for various game contexts that require deck-based operations.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Deck()

```
Deck::Deck (
            const std::string & deckName = "")  [inline]
```

Constructs a deck with an optional name.

Parameters

in	*deckName*	Optional name identifier for the deck.

Postcondition

Deck is initialized with an empty card list.

Here is the caller graph for this function:

5.8.3 Member Function Documentation

5.8.3.1 addCard()

```
void Deck::addCard (
            const Card & card)
```

Adds a card to the deck.

Adds a card directly to the deck vector.

Parameters

in	*card*	The card to be added.

Postcondition

The card is appended to the end of the internal vector.

Here is the caller graph for this function:

5.8.3.2 drawCard()

```
Card Deck::drawCard ()
```

Draws the topmost card from the deck.

Removes the last card from the internal vector (LIFO order) and returns it. If the deck is empty, an empty Card is returned instead.

Precondition

The deck should contain at least one card.

Postcondition

The deck's size decreases by one, unless empty.

Returns

The drawn Card object.

Warning

Returns a default-constructed Card if the deck is empty. Reshuffling functionality not implemented.

Returns the last element of the vector to simulate drawing from the top. Logs an error if called on an empty deck and returns a default card. Here is the caller graph for this function:

5.8.3.3 empty()

```
bool Deck::empty () const
```

Checks whether the deck is empty.

Returns true if there are no cards remaining in the deck.

Returns

True if the deck has no cards, false otherwise.

Postcondition

Does not modify the deck.

Here is the caller graph for this function:

5.8.3.4 loadFromJsonFile()

```
void Deck::loadFromJsonFile (
            const std::string & filename)
```

Loads cards into the deck from a JSON file.

The JSON file must contain an array of card definitions compatible with the Card class constructor. Each card may optionally define a `"copies"` field indicating how many duplicates to create.

Parameters

in	*filename*	The name or relative path of the JSON file to load.

Precondition

The file must exist and contain valid JSON.

Postcondition

The deck's card list is cleared and replaced with the loaded cards.

Exceptions

std::exception	If JSON parsing fails (caught internally and logged).

Warning

Logs errors to stderr if the file cannot be found or opened.

See also

Card::Card(const nlohmann::json&)

Uses PathUtils::getAssetPath() to resolve the full path of the file. Cards are loaded from JSON, supporting an optional "copies" field for duplication. Any parsing or I/O errors are printed to stderr.

Note

Default copies = 1 unless explicitly defined in JSON.

Here is the call graph for this function: Here is the caller graph for this function:

5.8.3.5 moveCardTo()

```
void Deck::moveCardTo (
            const Card & card,
            Deck & deck)
```

Transfers a card to another deck.

Parameters

in	*card*	The card to move.
out	*deck*	The destination deck that receives the card.

Postcondition

The specified card is added to the destination deck.

Note

This operation does not remove the card from the source deck.

Only appends the card to the destination deck — it does not remove it locally. This enables flexible gameplay behavior (e.g., cloning, discarding). Here is the call graph for this function:

5.8.3.6 shuffle()

```
void Deck::shuffle ()
```

Randomly shuffles the order of cards in the deck.

Uses a Mersenne Twister pseudo-random generator seeded from std::random_device. Suitable for gameplay randomness but not cryptographically secure.

Postcondition

All cards remain present, but in randomized order.

Uses std::shuffle with Mersenne Twister (mt19937) seeded from random_device. Ensures fair shuffling each time it's called. Here is the caller graph for this function:

5.8.3.7 size()

```
size_t Deck::size () const
```

Gets the number of cards currently in the deck.

Returns the current card count in the deck.

Returns

The count of cards.

Postcondition

Does not modify the deck.

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

- include/Deck.hpp
- src/Deck.cpp

5.9 Game Class Reference

Central game controller managing all gameplay mechanics and state.

```
#include <Game.hpp>
```

Public Member Functions

- Game (int boardSize, std::vector< Company > companyList)

  Constructs a new Game instance with specified board and company configuration.
- ~Game ()

  Destructs the Game instance, releasing dynamically allocated resources.
- void addPlayer (const std::string &name, Company *company)

  Registers a new player with their associated company.
- void setup ()

  Initializes the game board and deals starting cards.
- void mainLoop ()

  Enters the main game loop, processing events and rendering until window closes.
- const std::vector< Player > & getPlayers () const

  Retrieves the immutable list of all registered players.
- void startNewDay ()

  Executes start-of-day triggers for all played cards across all players.
- void drawCardForPlayer (Deck &deck, Player &player, int amount)

  Draws specified number of cards from a deck into a player's hand.
- void giveResourceToPlayer (int playerIndex, const std::string &resource, int amount, bool logToConsole=true)

  Grants a specified resource quantity to a player.
- bool spendResourceFromPlayer (int playerIndex, const std::string &resource, int amount, bool logTo↩ Console=true)

  Attempts to deduct a resource quantity from a player's inventory.
- void buildStage (int playerIndex, int x, int y, int z, const std::string &color)

  Constructs a stage (building) on a hex tile with specified color and ownership.
- void endTurn (bool logToConsole=true)

  Advances to the next player's turn, or starts a new day if all players finished.
- bool playCardForPlayer (int playerIndex, const std::string &cardName, bool logToConsole=true)

  Plays a card from a player's hand into their active play area.
- bool removePlayedCardForPlayer (int playerIndex, const std::string &cardName, bool logToConsole=true)

  Removes a card from a player's active play area (discards it).
- bool removeHeldCardForPlayer (int playerIndex, const std::string &cardName, bool logToConsole=true)

  Removes a card directly from a player's hand (discards without playing).
- int getCurrentDay ()

  Retrieves the current game day (round number).
- int getCurrentActivePlayerIndex ()

  Retrieves the index of the player whose turn is currently active.

5.9.1 Detailed Description

Central game controller managing all gameplay mechanics and state.

Game acts as the orchestrator for a turn-based hex board strategy game. It maintains:

- Game state (current day, active player)
- Player roster and their associated companies
- Hex board and tile ownership
- Card decks (draw and discard)

- Resource economy and transactions
- Command-based interface for game actions
- Rendering and event handling via SFML

The class follows a command pattern for user interactions, mapping string commands to handler functions that manipulate game state. This design allows for scripting, replay systems, and network play extensions.

Note

The game uses a -1 convention for playerIndex to indicate "current active player"

See also

Player, Board, Company, Deck

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Game()

```
Game::Game (
            int boardSize,
            std::vector< Company > companyList)
```

Constructs a new Game instance with specified board and company configuration.

Initializes the game environment including:

- Hexagonal board of given size
- SFML rendering window (800∗600)
- Font loading from assets
- Renderer and command console subsystems
- Company roster for player assignment
- Command handler registration

Parameters

in	*boardSize*	Radius of the hexagonal board (number of rings from center)
in	*companyList*	Vector of available companies for player assignment

Precondition

boardSize must be positive

companyList should not be empty for meaningful gameplay

Postcondition

Game object ready for player addition via addPlayer()

Window created but not yet displayed

Warning

Exits with error message if font file "consolas.ttf" cannot be loaded

See also

addPlayer(), setup()

Here is the call graph for this function:

5.9.2.2 ~Game()

```
Game::~Game ()
```

Destructs the Game instance, releasing dynamically allocated resources.

Ensures proper cleanup of renderer and console subsystems to prevent memory leaks.

Postcondition

All heap-allocated members freed

5.9.3 Member Function Documentation

5.9.3.1 addPlayer()

```
void Game::addPlayer (
            const std::string & name,
            Company * company)
```

Registers a new player with their associated company.

Adds a player to the game roster. Players are assigned turn order based on registration sequence. The company pointer establishes ownership relationships for tiles built by this player.

Parameters

in	*name*	Display name for the player
in	*company*	Pointer to the company this player represents (must outlive Game)

Precondition

company pointer must be valid and point to an existing Company object

Must be called before setup()

Postcondition

Player added to internal roster with zero-initialized resources

Note

Player index is implicitly assigned based on insertion order

See also

setup(), getCurrentActivePlayerIndex()

Here is the caller graph for this function:

5.9.3.2 buildStage()

```
void Game::buildStage (
            int playerIndex,
            int x,
            int y,
            int z,
            const std::string & color)
```

Constructs a stage (building) on a hex tile with specified color and ownership.

Changes tile ownership to the player's company and sets the tile's color. This is the primary mechanism for claiming territory and scoring points in the game.

Parameters

in	*playerIndex*	Index of the player performing the build action
in	*x*	X-coordinate of the target hex tile (cube coordinates)
in	*y*	Y-coordinate of the target hex tile (cube coordinates)
in	*z*	Z-coordinate of the target hex tile (cube coordinates)
in	*color*	Color name for the stage (must be valid per Colors::all)

Precondition

playerIndex must be valid (not -1)

Coordinates must reference an existing tile on the board

color must be a recognized color string

Postcondition

Tile ownership transferred to player's company

Tile color updated

Note

Does not validate color or coordinates; caller must ensure validity

See also

Board::setTileOwner(), Board::setTileColor()

5.9.3.3 drawCardForPlayer()

```
void Game::drawCardForPlayer (
            Deck & deck,
            Player & player,
            int amount)
```

Draws specified number of cards from a deck into a player's hand.

Transfers cards from the given deck to the player's held cards. Logs each draw to the console. Stops early if deck is exhausted.

Parameters

in,out	*deck*	Deck to draw from (modified by removing cards)
in,out	*player*	Player receiving the cards (hand is expanded)
in	*amount*	Number of cards to attempt drawing

Precondition

deck must be a valid Deck reference

player must be a valid Player reference

Postcondition

Up to 'amount' cards transferred from deck to player hand

Console messages logged for each drawn card

Note

If deck empties mid-draw, stops and logs a message

See also

Deck::drawCard(), Player::addHeldCard()

Here is the call graph for this function:

5.9.3.4 endTurn()

```
void Game::endTurn (
            bool logToConsole = true)
```

Advances to the next player's turn, or starts a new day if all players finished.

Increments the active player index. If the last player in turn order finishes, wraps to player 0 and increments the current day counter, triggering start-of-day effects.

Parameters

in	*logToConsole*	Whether to log turn transitions to console (default: true)

Postcondition

currentActivePlayerIndex advanced (or wrapped to 0)

If day boundary crossed, currentDay incremented and startNewDay() called

Console messages logged if logToConsole is true

See also

startNewDay(), getCurrentActivePlayerIndex()

Here is the call graph for this function:

5.9.3.5 getCurrentActivePlayerIndex()

```
int Game::getCurrentActivePlayerIndex ()
```

Retrieves the index of the player whose turn is currently active.

Returns

Integer index into the players vector (0-based)

See also

endTurn(), validateAndSetPlayerIndex()

5.9.3.6 getCurrentDay()

```
int Game::getCurrentDay ()
```

Retrieves the current game day (round number).

Returns

Integer representing the current day, starting from 0

See also

startNewDay(), endTurn()

5.9.3.7 getPlayers()

```
const std::vector< Player > & Game::getPlayers () const
```

Retrieves the immutable list of all registered players.

Provides read-only access to the player roster for querying game state.

Returns

Const reference to the internal player vector

See also

Player

5.9.3.8 giveResourceToPlayer()

```
void Game::giveResourceToPlayer (
            int playerIndex,
            const std::string & resource,
            int amount,
            bool logToConsole = true)
```

Grants a specified resource quantity to a player.

Increases the player's resource pool for the given resource type. If the resource doesn't exist in the player's inventory, initializes it to the given amount.

Parameters

in	*playerIndex*	Index of target player (-1 for current active player)
in	*resource*	Name of the resource (e.g., "gold", "wood", "influence")
in	*amount*	Quantity to add (can be negative to subtract)
in	*logToConsole*	Whether to print transaction to console (default: true)

Precondition

playerIndex must be valid or -1

Postcondition

Player's resource pool increased by amount

Console message logged if logToConsole is true

Note

Resources are string-keyed and dynamically created on first use

See also

spendResourceFromPlayer()

5.9.3.9 mainLoop()

```
void Game::mainLoop ()
```

Enters the main game loop, processing events and rendering until window closes.

Core game loop that:

1. Handles window events (close, input) via renderer
2. Processes queued console commands via executeCommand()
3. Renders the current game state (board, console output)

This loop runs indefinitely until the SFML window is closed by the user.

Precondition

setup() must have been called to initialize game state

Postcondition

Window closed and game terminated

Note

Blocks execution until game exit

See also

executeCommand(), Renderer::handleEvents(), Renderer::render()

Here is the caller graph for this function:

5.9.3.10 playCardForPlayer()

```
bool Game::playCardForPlayer (
            int playerIndex,
            const std::string & cardName,
            bool logToConsole = true)
```

Plays a card from a player's hand into their active play area.

Validates the player has the card in hand, transfers it to played cards, and executes the card's "onPlay" trigger immediately.

Parameters

in	*playerIndex*	Index of the player (-1 for current active player)
in	*cardName*	Exact name of the card to play
in	*logToConsole*	Whether to log action/errors to console (default: true)

Returns

true if card successfully played, false if not found in hand

Precondition

playerIndex must be valid or -1

Postcondition

If successful, card moved from hand to playedCards

Card's onPlay trigger executed

Console message logged if logToConsole is true

See also

removePlayedCardForPlayer(), Player::playCard()

Here is the call graph for this function:

5.9.3.11 removeHeldCardForPlayer()

```
bool Game::removeHeldCardForPlayer (
            int playerIndex,
            const std::string & cardName,
            bool logToConsole = true)
```

Removes a card directly from a player's hand (discards without playing).

Used for discard mechanics, hand management, or card destruction effects.

Parameters

in	*playerIndex*	Index of the player (-1 for current active player)
in	*cardName*	Exact name of the card to remove
in	*logToConsole*	Whether to log action/errors to console (default: true)

Returns

true if card found and removed, false if not in hand

Precondition

playerIndex must be valid or -1

Postcondition

If successful, card removed from heldCards

Console message logged if logToConsole is true

See also

playCardForPlayer(), Player::removeHeldCard()

Here is the call graph for this function:

5.9.3.12 removePlayedCardForPlayer()

```
bool Game::removePlayedCardForPlayer (
            int playerIndex,
            const std::string & cardName,
            bool logToConsole = true)
```

Removes a card from a player's active play area (discards it).

Used for card destruction, end-of-turn cleanup, or card replacement mechanics.

Parameters

in	*playerIndex*	Index of the player (-1 for current active player)
in	*cardName*	Exact name of the card to remove
in	*logToConsole*	Whether to log action/errors to console (default: true)

Returns

true if card found and removed, false if not in play

Precondition

playerIndex must be valid or -1

Postcondition

If successful, card removed from playedCards

Console message logged if logToConsole is true

See also

playCardForPlayer(), Player::removePlayedCard()

Here is the call graph for this function:

5.9.3.13 setup()

```
void Game::setup ()
```

Initializes the game board and deals starting cards.

Performs initial game setup:

- Randomly assigns colors and owners to half the board tiles (temporary)
- Sets remaining tiles to neutral (unowned)
- Loads card deck from "cards.json"
- Shuffles the draw deck
- Deals one card to the first player

This method implements the game's starting conditions as per design rules.

Precondition

At least one player must be added via addPlayer()

"cards.json" must exist and contain valid card definitions

Postcondition

Board tiles initialized with random colors and owners

Draw and discard decks created

First player has one card in hand

Warning

Neutral color (index 0 in Colors::all) is excluded from random selection

See also

addPlayer(), mainLoop()

Here is the call graph for this function: Here is the caller graph for this function:

5.9.3.14 spendResourceFromPlayer()

```
bool Game::spendResourceFromPlayer (
            int playerIndex,
            const std::string & resource,
            int amount,
            bool logToConsole = true)
```

Attempts to deduct a resource quantity from a player's inventory.

Validates that the player has sufficient resources before deducting. If insufficient, logs an error and returns false without modifying the player's inventory.

Parameters

in	*playerIndex*	Index of target player (-1 for current active player)
in	*resource*	Name of the resource to spend
in	*amount*	Quantity to deduct
in	*logToConsole*	Whether to print transaction/errors to console (default: true)

Returns

true if transaction succeeded, false if insufficient resources

Precondition

playerIndex must be valid or -1

Postcondition

If successful, player's resource reduced by amount

Console message logged if logToConsole is true

Warning

Returns false if resource doesn't exist in player's inventory

See also

giveResourceToPlayer()

5.9.3.15 startNewDay()

```
void Game::startNewDay ()
```

Executes start-of-day triggers for all played cards across all players.

Iterates through each player's active (played) cards and invokes their "onStartOfDay" trigger. This implements card effects that activate at the beginning of each game day (round).

Postcondition

All cards' onStartOfDay effects executed

Note

Called automatically when all players end their turn (day transition)

See also

endTurn(), Card::executeTrigger()

Here is the caller graph for this function:

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

- include/Game.hpp
- src/Game.cpp

5.10 GameConfig Struct Reference

Holds persistent player and company setup information.

Public Attributes

- int playerCount = 2
- std::vector< std::string > playerNames
- std::vector< std::string > companyNames
- std::vector< std::string > companySymbols

5.10.1 Detailed Description

Holds persistent player and company setup information.

5.10.2 Member Data Documentation

5.10.2.1 companyNames

```
std::vector<std::string> GameConfig::companyNames
```

Names of companies linked to players.

5.10.2.2 companySymbols

```
std::vector<std::string> GameConfig::companySymbols
```

Company symbols or abbreviations.

5.10.2.3 playerCount

```
int GameConfig::playerCount = 2
```

Number of players in the game (2-6).

5.10.2.4 playerNames

```
std::vector<std::string> GameConfig::playerNames
```

Names of all players.

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

- src/StartupMenu.cpp

5.11 Player Class Reference

Represents a player, managing their resources, cards, score, and associated company.

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

Collaboration diagram for Player:

Public Member Functions

- Player (const std::string &n, Company *c)
  Constructs a new Player.
- void addResource (const std::string &type, int amount)
  Adds a specified amount of a resource type.
- bool spendResource (const std::string &type, int amount)
  Attempts to spend a specified amount of a resource.
- int getResource (const std::string &type) const
  Retrieves the current amount of a resource.
- void addHeldCard (const Card &card)
  Adds a card to the player's hand.
- bool playCard (const std::string &cardName)
  Moves a card from hand to played cards.
- bool removePlayedCard (const std::string &cardName)
  Removes a played card by name.
- bool removeHeldCard (const std::string &cardName)
  Removes a held card by name.
- void addScore (int amount)
  Increases the player's score by a specified amount.
- void **resetScore** ()
  Resets the player's score to zero.
- void **printStatus** () const
  Prints a formatted summary of the player's current status to the console.

Public Attributes

- std::string **name**

  The player's display name.
- Company * **company**

  Pointer to the company this player is associated with.
- int **score** = 0

  Current score value.
- std::unordered_map< std::string, int > **resources**

  Map of resource type to quantity (e.g., "funds", "gear").
- std::vector< Card > **heldCards**

  Cards currently held in hand.
- std::vector< Card > **playedCards**

  Cards that have been played.

5.11.1 Detailed Description

Represents a player, managing their resources, cards, score, and associated company.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Player()

```
Player::Player (
            const std::string & n,
            Company * c)
```

Constructs a new Player.

Parameters

n	The player's name.
c	Pointer to the company associated with the player.

5.11.3 Member Function Documentation

5.11.3.1 addHeldCard()

```
void Player::addHeldCard (
            const Card & card)
```

Adds a card to the player's hand.

Parameters

card	The card to add.

Here is the caller graph for this function:

5.11.3.2 addResource()

```
void Player::addResource (
            const std::string & type,
            int amount)
```

Adds a specified amount of a resource type.

Parameters

type	The resource key.
amount	The quantity to add.

Here is the caller graph for this function:

5.11.3.3 addScore()

```
void Player::addScore (
            int amount)
```

Increases the player's score by a specified amount.

Parameters

amount	The number of points to add.

5.11.3.4 getResource()

```
int Player::getResource (
            const std::string & type) const
```

Retrieves the current amount of a resource.

Parameters

type	The resource key.

Returns

The quantity of the specified resource.

5.11.3.5 playCard()

```
bool Player::playCard (
            const std::string & cardName)
```

Moves a card from hand to played cards.

Parameters

cardName	The name of the card to play.

Returns

True if the card was found and played; false otherwise.

Here is the caller graph for this function:

5.11.3.6 removeHeldCard()

```
bool Player::removeHeldCard (
            const std::string & cardName)
```

Removes a held card by name.

Parameters

cardName	The name of the card to remove.

Returns

True if a card was removed; false otherwise.

Here is the caller graph for this function:

5.11.3.7 removePlayedCard()

```
bool Player::removePlayedCard (
            const std::string & cardName)
```

Removes a played card by name.

Parameters

cardName	The name of the card to remove.

Returns

True if a card was removed; false otherwise.

Here is the caller graph for this function:

5.11.3.8 spendResource()

```
bool Player::spendResource (
            const std::string & type,
            int amount)
```

Attempts to spend a specified amount of a resource.

Parameters

type	The resource key.
amount	The quantity to spend.

Returns

True if the player had enough resources; false otherwise.

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

- include/Player.hpp
- src/Player.cpp

5.12 Renderer Class Reference

Handles all visual rendering and window event management for the game.

`#include <Renderer.hpp>`

Public Member Functions

- Renderer (Board &board, sf::Font &font)

  Constructs a Renderer object.
- void handleEvents (sf::RenderWindow &window, CommandConsole &console)

  Processes SFML window events and forwards input to the console.
- void render (sf::RenderWindow &window, CommandConsole &console)

  Draws all visible elements, including the board and console.

5.12.1 Detailed Description

Handles all visual rendering and window event management for the game.

The Renderer is responsible for drawing tiles, board layout, and console output to the SFML window.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Renderer()

```
Renderer::Renderer (
            Board & board,
            sf::Font & font)
```

Constructs a Renderer object.

Parameters

board	Reference to the Board instance to render.
font	Reference to the font used for tile text and console output.

5.12.3 Member Function Documentation

5.12.3.1 handleEvents()

```
void Renderer::handleEvents (
            sf::RenderWindow & window,
            CommandConsole & console)
```

Processes SFML window events and forwards input to the console.

Parameters

window	The render window to poll for events.
console	Reference to the CommandConsole for input handling.

Here is the call graph for this function:

5.12.3.2 render()

```
void Renderer::render (
            sf::RenderWindow & window,
            CommandConsole & console)
```

Draws all visible elements, including the board and console.

Parameters

window	The render window where graphics are drawn.
console	Reference to the console for on-screen text rendering.

Here is the call graph for this function:

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

- include/Renderer.hpp
- src/Renderer.cpp

5.13 StartupMenu Class Reference

Manages the initial menu and configuration phase before the main game starts.

```
#include <StartupMenu.hpp>
```

Public Member Functions

- **StartupMenu** ()=default

  Default constructor.
- int StartMenuLoop ()

  Starts the main menu loop.

5.13.1 Detailed Description

Manages the initial menu and configuration phase before the main game starts.

Provides user interaction for choosing game settings, loading or saving configuration, and starting the gameplay session.

5.13.2 Member Function Documentation

5.13.2.1 StartMenuLoop()

```
int StartupMenu::StartMenuLoop ()
```

Starts the main menu loop.

Runs the startup menu loop for the game.

Displays menu options for starting the game, changing settings, or exiting the application. Handles configuration loading/saving and launches the game once setup is complete.

Returns

0 on successful completion or user exit.

Displays options to start the game, open settings, or exit. Loads saved configuration, sets up game data, and starts the main game loop.

Returns

0 when exiting the program.

Here is the call graph for this function:

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

- include/StartupMenu.hpp
- src/StartupMenu.cpp

5.14 Tile Class Reference

Represents a single board tile that can be owned by a company and display a color.

```
#include <Tile.hpp>
```

Public Member Functions

- **Tile** ()
  Constructs a neutral, unowned tile.
- void setOwner (Company *newOwner)
  Sets the owner of this tile.
- Company * getOwner () const
  Retrieves the company that owns this tile.
- const std::string & getColor () const
  Gets the tile's current color.
- void setColor (const std::string &newColor)
  Sets the tile's color.
- void **printInfo** () const
  Prints a formatted line describing ownership and color to the console.

5.14.1 Detailed Description

Represents a single board tile that can be owned by a company and display a color.

Each tile may have an owner (a pointer to a Company) and an associated color. Tiles start unowned and "neutral" in color by default.

5.14.2 Member Function Documentation

5.14.2.1 getColor()

```
const std::string & Tile::getColor () const
```

Gets the tile's current color.

Returns

Reference to the color string.

5.14.2.2 getOwner()

```
Company * Tile::getOwner () const
```

Retrieves the company that owns this tile.

Returns

Pointer to the owning company, or nullptr if unowned.

5.14.2.3 setColor()

```
void Tile::setColor (
            const std::string & newColor)
```

Sets the tile's color.

Parameters

newColor	New color name (string identifier).

Here is the caller graph for this function:

5.14.2.4 setOwner()

```
void Tile::setOwner (
            Company * newOwner)
```

Sets the owner of this tile.

Parameters

newOwner	Pointer to the company that now owns the tile. Can be nullptr for unowned.

Here is the caller graph for this function:

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

- include/Tile.hpp
- src/Tile.cpp

Chapter 6

File Documentation

6.1 include/Board.hpp File Reference

Declares the Board class which represents a hexagonal grid of tiles.

```
#include "CubeCoord.hpp"
#include "Tile.hpp"
#include "Company.hpp"
#include <unordered_map>
#include <vector>
```

Include dependency graph for Board.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Board

  Represents a hexagonal game board composed of cube-coordinate tiles.

6.1.1 Detailed Description

Declares the Board class which represents a hexagonal grid of tiles.

The Board class manages a collection of Tile objects arranged using cube coordinates. It provides functions for tile retrieval, neighbor lookup, ownership and color assignment, and overall grid initialization.

The cube coordinate system ensures consistent hex-grid relationships via the constraint: $x + y + z = 0$.

See also

CubeCoord, Tile, Company

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.2 Board.hpp

Go to the documentation of this file.

```
00001
00020
00021 #pragma once
00022 #include "CubeCoord.hpp"
00023 #include "Tile.hpp"
00024 #include "Company.hpp"
00025 #include <unordered_map>
00026 #include <vector>
00027
00040 class Board {
00041 public:
00049     Board(int radius);
00050
00058     std::unordered_map<CubeCoord, Tile, CubeCoordHash> tiles;
00059
00068     Tile* getTile(const CubeCoord& coord);
00069
00078     std::vector<CubeCoord> getNeighbors(const CubeCoord& coord) const;
00079
00091     void setTileOwner(int x, int y, int z, Company* company);
00092
00105     void setTileColor(int x, int y, int z, const std::string& color);
00106
00115     void printBoard() const;
00116
00117 private:
00119     int radius;
00120
00130     void generateBoard();
00131 };
```

6.3 include/Card.hpp File Reference

Declares the Card class which represents a singular card and its properties like name and description.

```
#include <string>
#include <vector>
#include <unordered_map>
#include <nlohmann/json.hpp>
```

Include dependency graph for Card.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Card

  Represents a playable card with triggers and associated actions.

6.3.1 Detailed Description

Declares the Card class which represents a singular card and its properties like name and description.

This class manages the individual cards that are loaded in a data driven manner from the cards.json asset. It stores information such as the cards name and description but additionally and more importantly it handles card trigger events that tie gameplay events to specific actions outlined on the card.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.4 Card.hpp

Go to the documentation of this file.

```
00001
00015
00016 #pragma once
00017 #include <string>
00018 #include <vector>
00019 #include <unordered_map>
00020 #include <nlohmann/json.hpp>
00021
00030 class Card {
00031 public:
00032     std::string name;
00033     std::string description;
00034
00047     std::unordered_map<std::string, std::vector<nlohmann::json» triggers;
00048
00052     Card() = default;
00053
00058     Card(const nlohmann::json& data);
00059
00065     void executeTrigger(const std::string& trigger, class Player& player) const;
00066 };
```

6.5 include/Colors.hpp File Reference

Declares the Colors class which stores the color standards.

```
#include <vector>
#include <string>
#include <algorithm>
#include <map>
#include <SFML/Graphics/Color.hpp>
```

Include dependency graph for Colors.hpp:

Classes

- class Colors

  Centralized static utility for color name validation and mapping.

6.5.1 Detailed Description

Declares the Colors class which stores the color standards.

Throughout the project seven standard colors and an additional eighth neutral color are commonly referencedand this class standarizes the use of the colors to allow for flexibilty if additional colors are added or removed later.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.6 Colors.hpp

Go to the documentation of this file.

```
00001
00015
00016 #pragma once
00017 #include <vector>
00018 #include <string>
00019 #include <algorithm>
00020 #include <map>
00021 #include <SFML/Graphics/Color.hpp>
00022
00030 class Colors {
00031 public:
00037     static const std::map<std::string, std::string> colorCodes;
00038
00042     static const std::map<std::string, sf::Color> sfmlColors;
00043
00047     static const std::vector<std::string> all;
00048
00054     static bool isValid(const std::string& color);
00055
00061     static const sf::Color& getSfmlColor(const std::string& color);
00062 };
```

6.7 include/CommandConsole.hpp File Reference

Declares the CommandConsole class which allows for user IO.

```
#include <SFML/Graphics.hpp>
#include <queue>
#include <vector>
#include <string>
#include "Board.hpp"
```

Include dependency graph for CommandConsole.hpp:

Classes

- class CommandConsole

  In-game text console handling user command input, output history, and paging.

6.7.1 Detailed Description

Declares the CommandConsole class which allows for user IO.

A key factor in the functionality of the program at whole is allowing the user to interact and receive output. This is done through the command console which only handles IO and passes all commands to the Game class to process.

See also

Board, Renderer, Game

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.8 CommandConsole.hpp

Go to the documentation of this file.

```
00001
00018
00019 #pragma once
00020 #include <SFML/Graphics.hpp>
00021 #include <queue>
00022 #include <vector>
00023 #include <string>
00024 #include "Board.hpp"
00025
00033 class CommandConsole {
00034 public:
00041     CommandConsole(Board& b, sf::Font& f, sf::Vector2f pos);
00042
00047     void draw(sf::RenderWindow& window);
00048
00053     void handleEvent(const sf::Event& e);
00054
00059     void print(const std::string& line);
00060
00065     void printPaged(const std::vector<std::string>& lines);
00066
00070     void showNextPage();
00071
00076     std::string nextCommand();
00077
00081     void clear();
00082
00087     bool hasCommand() const { return !pendingCommands.empty(); }
00088
00089     bool awaitingNextPage = false;
00090
00091 private:
00092     Board& board;
00093     sf::Font& font;
00094     sf::Text text;
00095     sf::Vector2f position;
00096
00097     std::string buffer;
00098     std::vector<std::string> outputLines;
00099     std::queue<std::string> pendingCommands;
00100
00101     // Paging
00102     std::vector<std::string> pagedBuffer;
00103     size_t pageIndex = 0;
00104
00105     // History
00106     std::vector<std::string> commandHistory;
00107     int historyIndex = -1;
00108
00109     // Config
00110     const size_t maxLines = 10;
00111 };
```

6.9 include/Company.hpp File Reference

Declares the Company class representing a business entity or player faction.

`#include <string>`
Include dependency graph for Company.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Company

  Represents a company, corporation, or in-game faction.

6.9.1 Detailed Description

Declares the Company class representing a business entity or player faction.

The Company class encapsulates the identifying information of a corporate or player-controlled entity within the game. Each company is characterized by a unique name and symbol which is currently just used for display.

See also

Tile, Board

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.10 Company.hpp

Go to the documentation of this file.

```
00001
00017
00018 #pragma once
00019 #include <string>
00020
00030 class Company {
00031 public:
00040     Company(const std::string& name, const std::string& symbol);
00041
00047     const std::string& getName() const;
00048
00054     const std::string& getSymbol() const;
00055
00062     void setName(const std::string& newName);
00063
00070     void setSymbol(const std::string& newSymbol);
00071
00072 private:
00074     std::string name;
00075
00077     std::string symbol;
00078 };
```

6.11 include/CubeCoord.hpp File Reference

Defines cube coordinates and hashing utilities for hex-grid operations.

`#include <functional>`
Include dependency graph for CubeCoord.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct CubeCoord

  Represents a position in a cube coordinate system.
- struct CubeCoordHash

  Hash functor for CubeCoord, allowing use in unordered containers.

6.11.1 Detailed Description

Defines cube coordinates and hashing utilities for hex-grid operations.

This file provides the CubeCoord structure used to represent positions in a 3D cube coordinate system (commonly used for hexagonal grids) and a custom hash functor to enable their use as keys in unordered containers.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.12 CubeCoord.hpp

Go to the documentation of this file.

```
00001
00015
00016 #pragma once
00017 #include <functional>
00018
00026 struct CubeCoord {
00027     int x;
00028     int y;
00029     int z;
00030
00037     CubeCoord(int x_, int y_, int z_) : x(x_), y(y_), z(z_) {}
00038
00044     bool operator==(const CubeCoord& other) const;
00045
00051     CubeCoord operator+(const CubeCoord& other) const;
00052 };
00053
00058 struct CubeCoordHash {
00064     std::size_t operator()(const CubeCoord& c) const noexcept;
00065 };
```

6.13 include/Deck.hpp File Reference

Declares the Deck class used for managing collections of Card objects.

```
#include <vector>
#include <string>
#include "Card.hpp"
```

Include dependency graph for Deck.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Deck

  Represents a collection of cards and provides utility operations for card management.

6.13.1 Detailed Description

Declares the Deck class used for managing collections of Card objects.

The Deck class represents a container of Card instances. It provides functionality for loading cards from JSON files, shuffling, drawing, transferring cards between decks, and querying deck state.

Note

This class assumes that the Card type supports construction from JSON objects.

See also

Card, PathUtils

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.14 Deck.hpp

Go to the documentation of this file.

```
00001
00018
00019 #pragma once
00020 #include <vector>
00021 #include <string>
00022 #include "Card.hpp"
00023
00034 class Deck {
00035 public:
00041     Deck(const std::string& deckName = "") : name(deckName) {}
00042
00044     std::string name;
00045
00047     std::vector<Card> cards;
00048
00064     void loadFromJsonFile(const std::string& filename);
00065
00071     void addCard(const Card& card);
00072
00085     Card drawCard();
00086
00095     void moveCardTo(const Card& card, Deck& deck);
00096
00106     void shuffle();
00107
00113     size_t size() const;
00114
00120     bool empty() const;
00121 };
```

6.15 include/Game.hpp File Reference

Core game engine managing turn-based hex board gameplay with card and resource mechanics.

```
#include <vector>
#include <string>
#include <functional>
#include <unordered_map>
#include <SFML/Graphics.hpp>
#include "Player.hpp"
#include "Board.hpp"
#include "Company.hpp"
#include "Deck.hpp"
```

Include dependency graph for Game.hpp:

Classes

- class Game

  Central game controller managing all gameplay mechanics and state.

6.15.1 Detailed Description

Core game engine managing turn-based hex board gameplay with card and resource mechanics.

The Game class orchestrates all gameplay systems including player management, turn sequencing, resource economy, card interactions, and command-driven game state manipulation. It integrates rendering, user input via console commands, and maintains the game loop.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.16 Game.hpp

Go to the documentation of this file.

```
00001
00015
00016 #pragma once
00017 #include <vector>
00018 #include <string>
00019 #include <functional>
00020 #include <unordered_map>
00021 #include <SFML/Graphics.hpp>
00022 #include "Player.hpp"
00023 #include "Board.hpp"
00024 #include "Company.hpp"
00025 #include "Deck.hpp"
00026
00027 class Renderer;
00028 class CommandConsole;
00029
00050 class Game {
00051 public:
00074     Game(int boardSize, std::vector<Company> companyList);
00075
00081     ~Game();
00082
00100     void addPlayer(const std::string& name, Company* company);
00101
00123     void setup();
00124
00141     void mainLoop();
00142
00149     const std::vector<Player>& getPlayers() const;
00150
00162     void startNewDay();
00163
00182     void drawCardForPlayer(Deck& deck, Player& player, int amount);
00183
00202     void giveResourceToPlayer(int playerIndex, const std::string& resource, int amount, bool
      logToConsole = true);
00203
00224     bool spendResourceFromPlayer(int playerIndex, const std::string& resource, int amount, bool
      logToConsole = true);
00225
00247     void buildStage(int playerIndex, int x, int y, int z, const std::string& color);
00248
00263     void endTurn(bool logToConsole = true);
00264
00284     bool playCardForPlayer(int playerIndex, const std::string& cardName, bool logToConsole = true);
00285
00303     bool removePlayedCardForPlayer(int playerIndex, const std::string& cardName, bool logToConsole =
      true);
00304
00322     bool removeHeldCardForPlayer(int playerIndex, const std::string& cardName, bool logToConsole =
      true);
00323
00329     int getCurrentDay();
00330
00336     int getCurrentActivePlayerIndex();
00337
00338 private:
00352     void executeCommand(const std::string& cmd);
00353
00363     void initializeCommandHandlers();
00364
00381     bool validateAndSetPlayerIndex(int& playerIndex, bool logToConsole = true);
00382
00400     bool parseCardNameWithOptionalPlayerIndex(std::istringstream& ss, std::string& cardName, int&
      playerIndex);
00401
00402     // ========================================
00403     // Command Handler Functions
00404     // ========================================
00405     // These private methods implement the logic for each console command.
00406     // They parse arguments from the input stream, validate inputs, and execute
00407     // the corresponding game action. Most log results to the console.
00408     // ========================================
00409
00416     void handleSetColor(std::istringstream& ss);
00417
00424     void handleSetOwner(std::istringstream& ss);
00425
00432     void handleBuild(std::istringstream& ss);
00433
00438     void handleListPlayers();
00439
```

```
00445     void handleShowResources(std::istringstream& ss);
00446
00452     void handleShowCards(std::istringstream& ss);
00453
00459     void handleGetCardCount(std::istringstream& ss);
00460
00467     void handleDrawCard(std::istringstream& ss);
00468
00475     void handleGiveResource(std::istringstream& ss);
00476
00483     void handleSpendResource(std::istringstream& ss);
00484
00491     void handlePlayCard(std::istringstream& ss);
00492
00499     void handleRemovePlayedCard(std::istringstream& ss);
00500
00507     void handleRemoveHeldCard(std::istringstream& ss);
00508
00513     void handleHelp();
00514
00515     // ========================================
00516     // Member Variables
00517     // ========================================
00518
00519     Board board;
00520     std::vector<Player> players;
00521     std::vector<Company> companies;
00522     std::vector<Deck> decks;
00523
00524     int currentDay = 0;
00525     int currentActivePlayerIndex = 0;
00526
00527     sf::Font font;
00528     sf::RenderWindow window;
00529     Renderer* renderer;
00530     CommandConsole* console;
00531
00537     std::unordered_map<std::string, std::function<void(std::istringstream&)>> commandHandlers;
00538
00545     Deck* getDeckByName(const std::string& deckName);
00546 };
```

6.17 include/PathUtils.hpp File Reference

Utility functions for locating executable, configuration, and asset directories.

```
#include <filesystem>
#include <string>
```

Include dependency graph for PathUtils.hpp:

Namespaces

- namespace PathUtils

  Contains helper functions for resolving application file paths.

Functions

- std::filesystem::path PathUtils::getExecutableDir ()

  Retrieves the directory where the executable is located.
- std::filesystem::path PathUtils::getConfigPath (const std::string &relativeFile="")

  Constructs a path within the "config" directory.
- std::filesystem::path PathUtils::getAssetPath (const std::string &relativeFile="")

  Constructs a path within the "assets" directory.

6.17.1 Detailed Description

Utility functions for locating executable, configuration, and asset directories.

Provides cross-platform path resolution for accessing the executable directory and related resource folders such as "config" and "assets".

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.18 PathUtils.hpp

Go to the documentation of this file.

```
00001
00014
00015 #pragma once
00016 #include <filesystem>
00017 #include <string>
00018
00023 namespace PathUtils {
00024
00029     std::filesystem::path getExecutableDir();
00030
00036     std::filesystem::path getConfigPath(const std::string& relativeFile = "");
00037
00043     std::filesystem::path getAssetPath(const std::string& relativeFile = "");
00044 }
```

6.19 include/Player.hpp File Reference

Defines the Player class, representing a participant with resources, cards, and a company affiliation.

```
#include <string>
#include <vector>
#include <unordered_map>
#include "Company.hpp"
#include "Card.hpp"
```

Include dependency graph for Player.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Player

  Represents a player, managing their resources, cards, score, and associated company.

6.19.1 Detailed Description

Defines the Player class, representing a participant with resources, cards, and a company affiliation.

Handles resource management, card play/removal, and scoring for an individual player in the game.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.20 Player.hpp

Go to the documentation of this file.

```
00001
00013
00014 #pragma once
00015 #include <string>
00016 #include <vector>
00017 #include <unordered_map>
00018 #include "Company.hpp"
00019 #include "Card.hpp"
00020
00025 class Player {
00026 public:
00028     std::string name;
00029
00031     Company* company;
00032
00034     int score = 0;
00035
00037     std::unordered_map<std::string, int> resources;
00038
00040     std::vector<Card> heldCards;
00041
00043     std::vector<Card> playedCards;
00044
00050     Player(const std::string& n, Company* c);
00051
00057     void addResource(const std::string& type, int amount);
00058
00065     bool spendResource(const std::string& type, int amount);
00066
00072     int getResource(const std::string& type) const;
00073
00078     void addHeldCard(const Card& card);
00079
00085     bool playCard(const std::string& cardName);
00086
00092     bool removePlayedCard(const std::string& cardName);
00093
00099     bool removeHeldCard(const std::string& cardName);
00100
00105     void addScore(int amount);
00106
00108     void resetScore();
00109
00111     void printStatus() const;
00112 };
```

6.21 include/Renderer.hpp File Reference

Declares the Renderer class responsible for drawing the game board and handling SFML rendering.

```
#include <SFML/Graphics.hpp>
#include "Board.hpp"
#include "Tile.hpp"
```

Include dependency graph for Renderer.hpp:

Classes

- class Renderer

  Handles all visual rendering and window event management for the game.

6.21.1 Detailed Description

Declares the Renderer class responsible for drawing the game board and handling SFML rendering.

Handles visual updates of the board, tiles, and console interface within the main SFML window.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.22 Renderer.hpp

Go to the documentation of this file.

```
00001
00013
00014 #pragma once
00015 #include <SFML/Graphics.hpp>
00016 #include "Board.hpp"
00017 #include "Tile.hpp"
00018
00019 class CommandConsole;
00020
00027 class Renderer {
00028 public:
00034     Renderer(Board& board, sf::Font& font);
00035
00041     void handleEvents(sf::RenderWindow& window, CommandConsole& console);
00042
00048     void render(sf::RenderWindow& window, CommandConsole& console);
00049
00050 private:
00055     void drawBoard(sf::RenderWindow& window);
00056
00058     Board& board;
00059
00061     sf::Font& font;
00062 };
```

6.23 include/StartupMenu.hpp File Reference

Declares the StartupMenu class for initializing and managing the game's main menu.

Classes

- class StartupMenu

  Manages the initial menu and configuration phase before the main game starts.

6.23.1 Detailed Description

Declares the StartupMenu class for initializing and managing the game's main menu.

Handles menu display, player and company setup, configuration persistence, and launching the main game loop.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.24 StartupMenu.hpp

Go to the documentation of this file.

```
00001
00014
00015 #pragma once
00016
00024 class StartupMenu
00025 {
00026 public:
00028     StartupMenu() = default;
00029
00039     int StartMenuLoop();
00040 };
```

6.25 include/Tile.hpp File Reference

Declares the Tile class, representing a single board tile that can be owned and colored.

```
#include <string>
#include "Company.hpp"
```

Include dependency graph for Tile.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Tile

  Represents a single board tile that can be owned by a company and display a color.

6.25.1 Detailed Description

Declares the Tile class, representing a single board tile that can be owned and colored.

Each Tile tracks its ownership by a Company and a color state, providing methods for setting and retrieving this information. Tiles default to being unowned and "neutral" in color.

Date

2025-11-06

Version

1.0

Author

Owen Chilson

6.26 Tile.hpp

Go to the documentation of this file.

```
00001
00014
00015 #pragma once
00016 #include <string>
00017 #include "Company.hpp"
00018
00026 class Tile {
00027 public:
00031     Tile();
00032
00037     void setOwner(Company* newOwner);
00038
00043     Company* getOwner() const;
00044
00049     const std::string& getColor() const;
00050
00055     void setColor(const std::string& newColor);
00056
00060     void printInfo() const;
00061
00062 private:
00063     Company* owner;
00064     std::string color;
00065 };
```