Carcassonne

Release -

Krzysztof Chorzempa

Jun 20, 2023

CONTENTS:

CHAPTER

ONE

CARCASSONNE DOCUMENTATION

1.1 logic package

1.1.1 Submodules

1.1.2 logic.const module

1.1.3 logic.feature module

class logic.feature.**Connection**(*side: int*, *number: int*)

Bases: object

Represents a connection (ability to connect to certain point on the tile) from a feature

Attributes

side
[int] Side, numbered from left side clockwise

number
[int] Number of the connection on the side (0 to 2)

rotate_once() → None
Rotate 90 degrees to the right

to_number() → int
Returns a number from 0 to 11 (beginning with left side at the bottom)

class logic.feature.**Feature**(*_type: int*, *connections: Sequence[*Connection*]*)

Bases: object

Represents a feature (farm, road, city, city with pennant or cloister)

Attributes

type
[int] Type of the feature

connections
[Sequence[Connection]] Connections (open "ports") to other features

meeple
[Player] Player that has a meeple on this feature (possibly None)

uuid
[UUID] Effect of wrong debugging, probably could be ommited

bindings
[Sequence[Feature]] The actual connections to the other features ("from this feature I *can get to* any other feature in this list")

parent_tile
[Tile] The tile this feature belongs to

scored
[bool] Whether it has been counted at the end (while counting open features)

`bind`(*feature:* Feature) → None
Binds to the given feature (makes it able to go from one to another)

`rotate`(*times: int*) → None
Rotate feature by 90 degrees clockwise *times* times

1.1.4 logic.game module

`class logic.game.CarcassonneGame`(*starting_tile:* Tile, *tileset: Sequence[*Tile*]*, *players: Sequence[*Player*]*)
Bases: `object`

Represents the whole game (game logic) Probably makes it possible to create multiple game states in one program

Attributes

tileset
[Sequence[Tile]] List of all tiles in the game except the starting one

players
[Sequence[Player]] List of players in the game

turn
[int] Index of the player whose turn is now

board
[dict] Dictionary mapping Coords to tiles, represents the actual board (only tiles that have been placed)

lastTile
[Tile] The last (most recent) tile placed in the game

phase
[int] 0 - placing a new tile 1 - placing a meeple (or decision not to place any)

scorer
[Scorer] Object used for routing on the features and dealing with score

tlesChanged
[set] A set of tiles that have changed in effect of placing the most recent meeple

classmethod from_file_and_names(*filename: str*, *playerNames: Sequence[str]*)
Creates a new instance of CarcassonneGame from a given tileset file and names of the players

Parameters

filename
[str] Filename of the file to read the tileset from

playerNames: Sequence[str]
Names of the players

get_current_player_color() → tuple

get_current_player_name() → str

get_current_tile() → *Tile*

get_winners() → list

handleEnd() → None
Score open features at the end of the game

is_finished() → bool

next_turn() → None
Go to the next turn

static parseFeatureText(*feature: str*) → *Feature*
Helper function to parse a single feature from a string

Parameters

feature
[str] String of the feature to parse

placeMeeple(*feature_index: int*) → None
Place a meeple

Parameters

feature_index
[int] Index of the feature on the tile to place the meeple on

place_tile(*coords:* Coords | *tuple[int, int]* | *list[int, int]*) → None
Place a tile

Parameters

coords
[Coords | tuple[int, int] | list[int, int]] Coords to place the tile on

1.1.5 logic.player module

class logic.player.**Player**(*name: str*, *color: tuple[int, int, int]*)

Bases: object

Represents a player

Attributes

name
[str] Name of the player

color
[tuple] Color of the player (color of the meeples)

score
[int] Current score

meeplesLeft
[int] How many meeples the player has left

addScore(*score_to_add: int*) → None
Updates the score

minusMeeple() → None
Subtract one meeple when placing it on the board

plusMeeple() → None
Add one meeple when a feature is completed

1.1.6 logic.scoring module

class logic.scoring.**Scorer**(*parent*)

Bases: object

calculate_points_for_closed(*feature:* Feature) → int
Calculate points for a closed feature (returns 0 for open features)

calculate_points_for_open(*feature:* Feature) → int
Calculates the points for an open feature (returns 0 for a closed feature)

check_any_meeples(*feature:* Feature) → bool
Check if there are any meeples on the feature

check_closed(*feature:* Feature) → bool
Check whether the feature is closed

count_closed_cities_near_farm(*feature:* Feature) → int
Counts the number of cities touching the farm

get_city_features_near(*farm_feature:* Feature) → list

Get city features on the same tile that are touching the given feature (farm)

get_connected_features(*feature:* Feature) → Sequence[*Feature*]

Get all features that can be reached from a given feature (including itself)

get_players_on_feature(*feature:* Feature) → Sequence[*Player*]

Get winning players on a given feature (with most meeples)

remove_meeples(*feature:* Feature) → set

Return meeples to their owners when a feature is completed

score_closed_feature(*feature:* Feature) → set

Calculate points for closed feature -> update scores -> remove meeples

score_open_feature(*feature:* Feature) → None

Calculate points for open features -> update scores -> remove meeples (just in case)

1.1.7 logic.tile module

class logic.tile.**Tile**(*game:* CarcassonneGame, *features: Sequence[*Feature*]*)

Bases: object

Attributes

game

[CarcassonneGame] Game that this tile belongs to (doesn't have to be on the board yet)

features

[Sequence[Feature]] Features that this tile consists of

uuid

[UUID] Again, wrong debugging, probably could be omitted

coords

[Coords] The coords of the tile on the board

ensureCorrect() → None

Ensure that the tile is correct (features collectively have 12 unique connections)

get_feature_by_connection(*connection:* Connection) → *Feature*

Return a feature belonging to this tile with the given connection

placeMeeple(*ind*, *player*) → None

Place a meeple of the given player on the feature with index *ind*

rotate(*times: int*) → None

Rotate the tile by 90 degrees clockwise *times* times

1.1.8 logic.utils module

class logic.utils.**Coords**(*x: int*, *y: int*)

Bases: object

Class to manage coordinates on a board

get_adjacent_coords() → Sequence[*Coords*]

Returns a list of adjacent coords (without itself)

get_coords_around() → Sequence[*Coords*]

Returns a list of coords around and itself (to manage cloisters)

to_tuple() → tuple

Returns a tuple representation (x, y)

logic.utils.**get_side_conn_list**(*tile*, *side*) → list

Get a list of feature types with connections on the given side of the given tile (used for checking if a tile can be placed)

logic.utils.**invert_side**(*side: int*) → int

Convert a side to its opposite (left-right, top-bottom)

logic.utils.**is_nearby_connection**(*conn1*, *conn2*) → bool

Check if feature with *conn1* connection is touching a feature with *conn2* connection

logic.utils.**is_nearby_feature**(*f1:* Feature, *f2:* Feature) → bool

Check if two features are touching each other

logic.utils.**parse_connection_number**(*connection_number: int*) → *Connection*

Make a Connection object out of a number from 0 to 11

1.1.9 Module contents

1.2 main module

main.**main**()

1.3 view package

1.3.1 Submodules

1.3.2 view.const module

1.3.3 view.game module

class view.game.**GameView**

Bases: object

Manages the whole graphics of the game

nextScene() → None

run() → None

1.3.4 view.scenes module

class view.scenes.**EndScene**(*parent*)

Bases: *Scene*

Scene for displaying the winners

draw() → None

process_events(*event*) → None

setup() → None

Setup the scene (for things that cannot be accessed at the start of the program)

class view.scenes.**GameScene**(*parent*)

Bases: *Scene*

Scene for the actual gameplay

draw() → None

handleEnd() → bool

Handle the game end: tell the game logic and hide ui elements

process_events(*event*) → None

setup() → None

Setup the scene (for things that cannot be accessed at the start of the program)

class view.scenes.**Scene**(*parent*, *background_color*)

Bases: object

clear() → None

Draw background on top to clear the screen

abstract draw() → None

abstract process_events(*event*) → None

abstract setup() → None

Setup the scene (for things that cannot be accessed at the start of the program)

class view.scenes.**WelcomeScene**(*parent*)

Bases: *Scene*

Scene for inputting the number of players and their names

draw() → None

process_events(*event*) → None

setup() → None

Setup the scene (for things that cannot be accessed at the start of the program)

validateNames() → bool

Validate the given names of the players (non-empty, unique)

validateNumber() → bool

Validate the given number of players

1.3.5 view.ui_widgets module

class view.ui_widgets.**BoardWidget**(*parent:* Scene, *tile_size: int*)

Bases: *UIWidget*

Widget containing tiles on the board and handling the placing of tiles and meeples (basically coordinating the Tile widgets)

draw() → None

on_resize() → None

Adjust to the new size of the screen

set_tile_to_place(*tile_to_place*) → None

update_tile(*new_tile:* Tile, *pos: tuple[int, int]*) → None

class view.ui_widgets.**InfoWidget**(*parent*, *players*)

Bases: *UIWidget*

Widget for displaying information about players

draw() → None

hide() → None

class view.ui_widgets.**PlayerWidget**(*parent*, *player*, *pos*)

Bases: *UIWidget*

Widget for displaying information about a single player

draw() → None

hide() → None

render() → None

set_label() → None

class view.ui_widgets.**TextWidget**(*parent:* Scene, *text: str*, *fontsize: int*, *pos: tuple[int, int]*, *color: tuple[int, int, int]*)

Bases: *UIWidget*

A redundant widget that can be replaced by pygame_gui's UILabel

draw() → None

class view.ui_widgets.**TileWidget**(*tile*, *pos*, *size*, *parent*)

Bases: *UIWidget*

Tile widget

draw() → None

on_resize() → None

Adjust to the new size of the screen

render() → None

Draw the tile image and remember it

class view.ui_widgets.**UIWidget**(*parent:* Scene, *pos: tuple[int, int]*)

Bases: object

Abstract class for custom UI widgets

abstract draw() → None

get_screen()

Get the screen that this feature should be drawn on

on_resize() → None

Adjust to the new size of the screen

1.3.6 view.utils module

view.utils.**alignMousePosition**(*mousePosition*, *screenSize*, *tileSize*) → tuple[int, int]

Convert the mouse position to coordinates of a tile on board

view.utils.**deCornify**(*pos: tuple[float, float]*, *tilesize: int*) → tuple[float, float]

In fact: deEgify. Used to move a meeple away from the edge of a tile (to be visible as a whole)

1.3.7 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

m

v

INDEX

V

W