SWE-Skat

version

Pascal Stehling

Februar 09, 2020

Contents

Welcome to SWE-Skat's documentation!

Code API List

Here can you find all API-References for the Classes which were used in this Project.

Bidding Functions

This File Contains the Bidding Class. Its neccessary to play the Bidding Phase

class `modules.Bidding.`**`Bidding`** (`settings`, `players`)
The Bidding Class contains all Functions relevant for Bidding.

`end_bidding` **()**
Ends the bidding Phase, if there is an bid Player, he won the bidding, else it starts new with new cards

Returns: Tuple with the the player who has the next turn and the new gamstate (turn, gamestate)
Return type: tuple

`get_new_turn` **()**
Get the new turn for the next round of bidding

`is_bidding_over` **()**
Checks if the bidding finished

Returns: True if the bidding is finished, else False
Return type: bool

`make_bid` **()**
Main Function for bidding. Ask Player if he wants to bid, checks if bidding Phase ends and select the new turn

`play_bidding` **()**
This function is the main Function of this Class. With this the Bidding Phase can be Played

Returns: The Player who won Bidding or None if all passed and the new Gamestate
Return type: tuple

Card Functions

This File Contains the Card Class. Each Card Object is a single Card which can be Played

class `modules.Card.`**`Card`** (`value`, `suit`)
Creates a card object. This has a suit as string, as Unicode character and the value of the suit. It also includes the value of the card (7-10, J, Q, K ,A) and the points of the card value.

Returns: A Card Object
Return type: card

`equal_suit` **(**`other_card`**)**
Checks if the Card has the same suit as another Card

Parameters: **other_card** (*Card*) – Card to check if equal suit
Raises: **TypeError** – if other_card is not of Type Card
Returns: True if the Cards have the same suit
Return type: bool

`get_ascii_card` **()**
Returns the Card as Ascii Art, where each element of the list is one line of the picture.

Returns: list with the elements of the Ascii Art Card
Return type: list

get_card_tuple ()
Get the suit value and the card value

Returns: tuple with suit value and card value
Return type: tuple

has_higher_suit_val (other_card)
Checks if the card is higher than other_card

Parameters: **other_card** (*Card*) – Card to check with
Returns: True if the main Card (self) is higher, False if other_card is higher
Return type: bool

has_higher_value (other_card)
Checks if the main card (self) has an higher value than other_card

Parameters: **other_card** (*Card*) – Card to check with
Returns: True if the main card (self) is higher
Return type: bool

ishigher (other_card, check_suit_val=False)
Checks if the card is higher than other_card

Parameters:
- **other_card** (*Card*) – Card to check with
- **trumpf** (*str*) – Trumpf which is played at the moment
- **order_dict** (*dict*) – dictionary with the ranking order of the cards
- **check_suit_val** (*bool, optional*) – If the higher suit wins, if no card is Trumpf and there are not equal suit. If False the main card (self) is winning. Defaults to False.

Raises: **TypeError** – if other Card is not of Type Card
Returns: True if the main Card (self) is higher, False if other_card is higher
Return type: bool

istrumpf ()
Checks if the Card is Trumpf

Parameters: **trumpf** (*str*) – the trumpf which is played at the moment
Raises: **TypeError** – if trumpf not None or not string
Returns: True if card is trumpf
Return type: bool

print ()
Prints the Ascii Art Card

same_suit_or_trumpf (other_card)
Check if 2 Cards have the same Suit or both are trumpf. :returns: True If the 2 Cards have the same suit or are both trumpf, else False :rtype: boolean

class modules.Card.**EmptyCard**
Creates an Empty card, with no text

Cards Functions

This File contains the Cards object. This is a container CLass for Multible Card Objects

class `modules.Cards.`**`Cards`** (`settingContainer`, `cards=None`)
Creates a Cards object, which is a container for multible Card objects. It has functions to manipulate multible Card objects.

`add_card` **(**`card`**)**
Add a new Card to the Cards object

Parameters: **card** (*Card*) – The Card that should be added

`add_card_and_sort` **(**`card`**)**
Add a new Card to the List and sort the List afterwards

Parameters: **card** (*Card*) – The new Card the should be Added

`create_shuffled_cards` **()**
Creates 32 cards and returns them shuffled

`empty_cards` **()**
Remove all Cards from the Cards Object

`get_jacks` **()**
Get all Jacks from the Cards Object

Returns: A List with all Jack Card objects
Return type: List

`index` **(**`card`**)**
Reeturns the Index of the Card in the list of all Cards

Parameters: **card** (*Card*) – The Card, where the Index should be found
Returns: The index of the Card in the Cards Object list of all Cards
Return type: int

`print_cards_ascii` **(**`card_delimiter=':'`**)**
Prints all Card objects as Ascii-Art in the Terminal.

Parameters: **card_delimiter** (*str, optional*) – The Symbols which are used between 2 Cards. Defaults to ":".

`remove` **(**`card_to_remove`**)**
Remove a Card from the List

Parameters: **card_to_remove** (*Card*) – The Card the should be removed

`sort_cards` **()**
Sorts all Card objects with a simple Bubble sort. There can only be 12 Cards at your Hand, so Performance isnt that important

Player Functions

This File Contains the Player Class which has all Informations about a single Player

class `modules.Player.`**`Player`** (`player_name`, `**settings`)
This Class contains all Infomations about a Player

`has_cards` **()**
Checks if the Player has Cards

Returns: True if the Player has Cards, else False
Return type: bool

Players Functions

This File contains the Player CLass. which is a container for Player objects

class `modules.Players.`**`Players`**(`settingContainer`, `**kwargs`)
The Player Class is an container for multible Player Objects

`get_next_player`(`player`)
Get the next Player in the List, if the list is at the End, it starts from the beginning

Parameters:	**player** (*Player*) – The Player at the moment
Returns:	the Player, after the Player which was given as input
Return type:	Player

`get_player_by_num`(`num`)
Get a Player by its num which was assigned when creating a Player object

Parameters:	**num** (*Int*) – The num of the Player
Raises:	**Exception** – If a Player with this num doesen't exist
Returns:	The Player with the corresponding num
Return type:	Player

`reset`()
Reset the Player for a new Round.

`set_players_on_next_position`()
Change the positions of all Players. Forhand –> Middlehand, Middlehand –> Backhand, Backhand –> Forhand

`sort_cards`()
Sort all Cards of all Players

Round Functions

The File with the Round Class

class `modules.Round.`**`Round`**(`players`, `settingContainer`)
The Round Class has all Functions for Playing and Evaluating a Played Round

`end_round`()
Calculates all Points and ends the played round

Returns:	returns itself
Return type:	Round

`give_cards`()
give Cards to every Player and put 2 into the Skat

`play_round`()
Play a round

Returns:	return its own object
Return type:	Rround

`set_card_default_values`()
sets the static Values of Card to its default

`set_gamemode`()
Ask the SIngle Player which play Types he wants to chose and sets it

`setup`()

Starts the Setup, in which the single Player can take the Skat and choose the gamemode

Returns: returns itself
Return type: Round

start_bidding ()
Start and play the bidding phase

Returns: the new gamstate after the bidding Phase
Return type: int

start_new_round ()
Starts a new round

SettingContainer Functions

This File contains the SettingContainer Class

class modules.SettingContainer.**SettingContainer** (setting_dict)
This Class contains all Settings which are relevant for the Game.

***static* create_SettingContainer_from_file (language='en')**
creates a SettingContainer Object from a setting.json file

Parameters: **language** (*str, optional*) – the Language of the game. Defaults to "en".
Returns: returns a SettingContainer Object
Return type: SettingContainer

get_sorted_suit_list ()
returns a sorted list of all suits

Returns: the List of string with the suits
Return type: List

Stich Functions

This File Contains the Stich Class

class modules.Stich.**Stich** (players, turn, settings)
The Stich CLass contains all functions which are neccessary to play a single Stich

assign_stich_to_winner ()
assign the Cards of the Stich to the winner

Returns: retunrs itself
Return type: Stich

get_winner ()
Gets the cards that were Played and returns the number of the Player who won

play_stich ()
Play a single Stich

Returns: returns itself
Return type: Stich

print_card_table ()
Prints the Cards that were Played plus empty cards to show how many need to be played

Tools Functions

Here are functions witch are needed in diffrent states of this programm

`modules.tools.`**`get_user_card`** (`show_message`, `error_message`, `user_cards`)
Get a Card from a Userinput

`modules.tools.`**`get_user_true_false`** (`show_message`, `error_message`, `cards`)
Get True or False if user wants to take the bid

`modules.tools.`**`user_select_card`** (`show_message`, `error_message`, `user_cards`)
Letthe User select a Card from ts Cards

Parameters:
- **show_message** (*Str*) – Message whitch will be shown befor selecting
- **error_message** (*str*) – Message that will be shown if an error occures
- **user_cards** (*Cards*) – the Cards of the user

Returns: returns the new User Cards without the selected card and the selected card

Return type: Tuple

Metrics

In this project 2 different software solutions were used to create different metrics.

The first tool used was **SonarCloud**, which is the cloud solution of **SonarQube**. The dashboard can be found at https://sonarcloud.io/dashboard?id=PascalStehling_Skat Metrics that were especially important is:

- *Duplications*: Percentage of all rows, how many of them are duplicates
- *Coverage*: percentage of how many lines of code are covered by tests.

Other metrics are also created by **SonarQube**, such as *security*, *reliability* and *maintainability*.

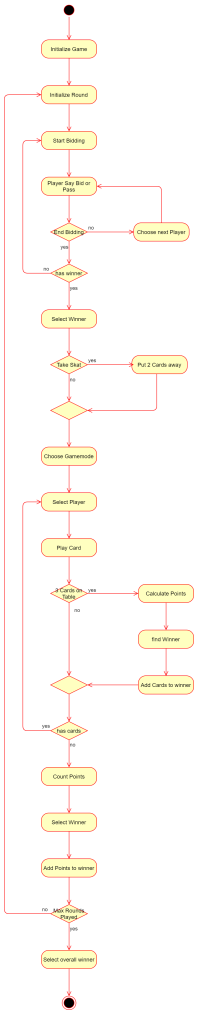
As second tool **Codacy** was used. This is also an online solution that can be easily linked to github accounts. The setup and usage is more user-friendly than **SonarQube**. The dashboard can be accessed via https://app.codacy.com/manual/PascalStehling/Skat/dashboard. **Codacy** can also calculate the metrics *Duplications* and *Coverage*. But there are also other metrics like *percentage of errors in code* and *percentage of complex files*.

All in all, I think **Codacy** is better than **SonarQube** because it is easier to get started and settings can be made more easily via the website.

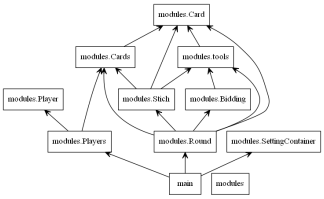
UML Diagrams

To better understand the code and the basic concept, 3 UML diagrams were created.

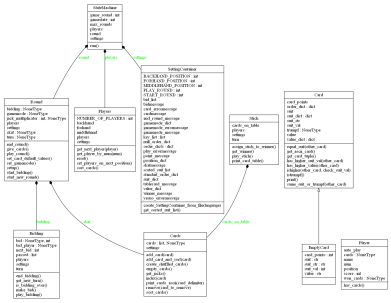
The first is an activity diagram, which shows the basic flow of a skat game.



The second one is a package diagram and shows which files call each other and are connected.



The third diagram is a class diagram that shows the general structure of the classes and also some connections between them.



Build Management

The main build management tool used was Bazel. With this tool it is possible to build a Skat.exe and execute the tests. To do this, the command *bazel build Skat* can be executed in the root folder. This will create a Skat.exe in the bazel-bin folder. To run the tests, the command *bazel run test* can be used. For more information on installing Bazel, please visit their website.

For further automation the tool doit can also be used. With this tool all unit tests can be executed with the command *doit run_test*. In addition, a coverage.xml is created which contains all information about the coverage of the code. Furthermore, with the command *doit lint_modules* all files in the folder modules can be viewed with pylint. With the command *doit bazel_build* the Skat.exe can be built with bazel. To use doit you just have to install all the required libraries with *pip install -r requirements.txt*.

This is my small Pet project for Softwwarengeneering. I tried to build a small Skat game which can be played in the command line.

Indices and tables

- **genindex**
- **modindex**
- **search**

Index

U

Python Module Index

m