卒業研究報告書

題目

Python を用いた麻雀における最下位にならない AI の開発

指 導 教 員

石水 隆 講師

報告者

17-1-037-0219

富田零生

近畿大学理工学部情報学科

令和 3 年 1 月 28 日提出

概要

麻雀には順位ウマというものがあり, 対局終了時に 1 位が大勝ちして, 4 位が大負けするルールがある. そのため順位ウマがある麻雀では 1 位を狙い, 4 位になることを避けなければいけない. 麻雀はツモ牌や配牌といった不確定要素によって 1 試合に何度も和了できたり, 逆に 1 度も和了できなかったりする. したがって, 麻雀で 1 位を安定して目指すことは難しい. しかし, 麻雀は相手の捨牌などを見てできるだけ放銃を避けて 4 位になりにくくなることはできる. そこで, 本研究では守りに重点を置き, できるだけ 4 位を取らないような麻雀 AI の開発を目指す. 本研究では様々な状況で和了を目指すかオリるかを決める条件を変えながら麻雀 AI 同士を対戦させ, どの条件が最も 4 位を回避できているかを検証する.

目次

1　序論

1.1　本研究の背景

1.2　既存の麻雀プログラム

1.3　本研究の目的

1.4　本研究の構成

2　麻雀について

2.1　麻雀のルール

2.2　フリテンについて

2.3　麻雀の既知の戦略

3　麻雀プログラム

4　研究内容

5　実験結果・統計的検証および考察

5.1　実験結果

5.2　統計的検証および考察

6　結論・今後の課題

1. 序論

1.1. 本研究の背景

4人打ちでの麻雀には，対局終了時に最終順位に応じたポイントを支払い，受け取りをする順位ウマというものがある．このルールは麻雀のプロリーグである「Mリーグ」[1]や多くのネット麻雀の段位戦などで採用されている．順位ウマは4位が1位に，3位が2位にポイントを支払いする仕組みである．そのため対局終了時に1位と2位のポイントはウマの分増え，3位と4位のポイントはウマの分減る．例えばウマが 10-20 であったとすると3位から2位へ 10000 点支払い，4位から1位へ 20000 点支払う．この順位ウマがあることで順位自体に価値を持たせて，対局終了時に点数が僅差でも 1 位と 4 位に大きく差が生まれるようになり，持ち点を増やすだけでなく高順位を狙って試合を進めることになる．このように順位ウマのある麻雀は，できるだけ1位を狙い，なるべく4位を避けなければいけないゲームであることがわかる．

麻雀は零和有限不確定非完全情報ゲームに分類される．不確定とは，運によって勝敗が左右されることであり，非完全情報ゲームとは相手の手牌や，残りの山の牌，相手の行動の選択が不明であることである．したがって，麻雀は相手の手牌が不明であることや，運によって勝敗が大きく左右されることから完全解析が難しいとされている．このため，最適解とされる行動を取っても必ずしも勝てるとは限らない．したがって，捨て牌や鳴き牌，自身の経験を頼りに必要とする条件を与えて試行錯誤して勝率を上げるしかない．

日本の麻雀のルールには自摸和した場合は他家が分担して点を支払うのに対して，自分の捨牌で他家を栄和させてしまった場合には 1 人で点を支払わなければならない．したがって，自分の点を減らさないためには放銃が避けることが重要である．

1.2. 既存の麻雀プログラム

麻雀は不確定ゲームかつ非完全情報ゲームであるため，良い手の判定が難しく，将棋や囲碁と比べると強い麻雀プログラムの作成は困難である．初期の麻雀プログラムでは，他家の手牌が見えないことを利用して他家は配牌時に一向聴となるようにし，聴牌時以外は全て自摸切りし聴牌すればリーチ，というものがよく用いられた．当然これはイカサマであり，麻雀 AI と呼べるものではない．

麻雀プログラムを作る上で比較的有望なのが，手牌および相手の捨牌から各牌の残り枚数を求め，最も和了率が高くなるように捨牌を選ぶ手法である．この手法は平均的にはそれなりの和了率が得られる一方，放銃を考えていないため高得点を振り込んでしまうことも多い．

また，昨今では，機械学習によるゲーム AI が注目されており，将棋や囲碁ではプロに勝つものも現れている．しかし，プロ棋士の棋譜を学習データとして用いることができる将棋や囲碁とは違い，適当な学習データが存在しない麻雀では機械学習も難しい．とは言え，昨今では不確定非完全情報ゲームに対する AI 開発も進んでおり，ある程度の強さを持つ麻雀 AI も存在している．現在，強い麻雀 AI としてはマイクロソフトの「Microsoft Suphx」」がある．この麻雀 AI は日本のオンライン麻雀対戦プラットフォーム「天鳳」において AI として初めて 10 段を達成し，その強さは，トッププレイヤーに匹敵している．[4][5]

1.3. 本研究の目的

1.1 節で述べた通り，自分の捨牌で他家を栄和させてしまった場合，1人で点を支払わなければならない．したがって，麻雀は基本振り込まないことが大事である．他家が自摸和した場合は点を分担して支払うため，放銃するよりは少なく済み，最後まで振り込まずに流局させればノーテン罰符の安い点数で済ませられる．そこで本研究では守りに重点を置き，聴牌したなら和了を目指し，聴牌前なら様々な条件で勝負するか降りるかを判断するべきなのかを調べ，順位ウマありの麻雀でで

きるだけ4位を取らないような麻雀 AI の開発を目指す．また，開発した AI を用いて4人麻雀の半荘戦での最終順位を調査することで，最下位を回避できているかを検証する．

1.4. 本報告書の構成

本報告書の構成を以下に述べる． 2 章では麻雀ゲームについての簡単な概要を説明し, 3 章では作成した麻雀のプログラムと AI について述べ, 4 章では作成した AI の対戦結果を述べる. 5 章ではこれまでに得られた結果から今後の課題について述べる．

2. 麻雀について

本章では麻雀のルールについて以下に述べる.

2.1. 麻雀ゲームの概要

麻雀は 3〜4 人で行うゲームである.本研究では 4 人で行うものとして述べる.麻雀は 34 種の牌をそれぞ れ 4 枚づつの全 136 枚の牌を使用する.

まず最初にプレイヤーの中から親を決める．そして，時計回りランダムに 13 枚の牌(以下，配牌とする)が 配られ，残りの牌は山として置かれる.その後親から順番に，1 枚山から牌を引き(以下，ツモとする)，ツモ した牌を含めた 14 枚の手牌から 1 枚捨てる行為を反時計回りに繰り返す.これらの中で他プレイヤーより早く決められた役を目指す．和了するには和了形を作る必要があり，麻雀の和了形は，雀頭と面子 4 組からなる基本形と七対子および国士無双の特殊形がある．基本形の雀頭は 2 枚 1 組の対子 1 つであり，面子は刻子，順子，槓子のいずれか 3 枚 1 組の組み合わせである.

ツモした後の 14 枚の手配で和了形が完成した場合「自摸和」を宣言する.また，自分以外のプレイヤーの手番中で，手番 プレイヤーが捨てた牌 1 枚と自分の手牌 13 枚を合わせた 14 枚で和了形ができる場合「栄和」を宣言する.このとき手番プレイヤーが和了牌を捨てることを「放銃」という.

和了形に含まれる役の組み合わせにより得点が決まっておりそれに沿って点数のやり取りを行う.自摸和した場合，和了したプレイヤーに対して他のプレイヤーが分担して得点を支払う．一方，栄和したときは，和了したプレイヤーに対して，放銃したプレイヤーが全ての得点を支払う．親が和了した場合，得られる点数は 1.5 倍になる.親の自摸和では他のプレイヤーは得点の 0.5 倍ずつ支払い，栄和では放銃したプレイヤーが得点の 1.5 倍を支払う．一方，子の自摸和では親は得点の 1/2 を，残りの子は得点の 1/4 を支払い，栄和では親子に関係なく放銃したプレイヤーが得点を全額支払う．配牌から和了までの一連の流れを 1 局という．局が終われば，全ての牌をシャッフルし直し，次の局を始める.もし山から引ける牌が無くなっても誰も和了できなかった場合はその時点でその局は終わり，同様に全ての牌をシャッフルし直し，次局を始める.親が和了した場合は和了したプレイヤーは親を継続し，子が和了した場合は反時計回りに親を移動する.流局した場 合は，親が後一枚の牌和了できる場合(以下，テンパイとする)，親が継続する.そうでない場合(以下，ノーテンとする)は親を移動する.また，ノーテンの場合はペナルティとして点数を支払わなければならない.

通常，麻雀は半荘と呼ばれる 8 局（全員が 2 回親になるまで，ゲーム過程により増減する）を 1 試合とする.半荘終了後，最終的に各プレイヤーの持ち点の過多により勝敗を決める.従って，プレイヤーの目指すべき事 は如何に高い点数を得るか，如何に相手へ点数を渡さないかに尽きる.

2.2. フリテンについて

ここでは麻雀のルール「フリテン」について説明する．フリテンとは，他プレイヤーから栄和ができないことを指す．以下のうちどれかに当てはまればフリテンになる.

- 和了牌を自分が捨てている
- 自分が牌を捨ててから次のツモまでに和了牌が捨てられている
- 和了牌をリーチ後に見逃した

1 つ目の「和了牌を自分が捨てている」は和了牌が複数あっても，どれか 1 つ捨てているとフリテンとなる.

2.3. 麻雀に関する既知の戦略

本節では，麻雀に関する既知の戦略について述べる．

1.1 節で述べた通り，日本の麻雀のルールには自摸和した場合は他家が分担して点を支払うのに対して，自分の捨牌で他家を栄和させてしまった場合には 1 人で点を支払わなければならない．したがって，自分の点を減らさないためには放銃が避けることが重要である．

他家がすでに捨てた牌と同じ牌を捨てた場合，フリテンのルールによりその牌で放銃することはない．このため，他家がすでに捨てた牌は，「現物」と呼ばれ安全な牌となる．そこで，オリる場合に「現物」を捨て続ける「現物切り」と呼ばれる戦略がある．また，麻雀では，例えば 2, 3 を持っている場合に 1，4 の両面待ちにすることが多い．フリテンのルールでは，2，3 を持っている時に 4 をすでに捨てている場合，4 だけでなく 1 もフリテンとなる．よって「1，4，7」「2，5，8」「3，6，9」の組み合わせは「筋」と呼ばれ，ある牌が捨てられたときにその筋の牌は安全である確率が高いため，筋の牌を捨てる「筋切り」と呼ばれる戦略がある．

聴牌したときにリーチすると和了時に得られる点が増えるが，他家に聴牌したことを知られてしまうこと，手牌の組み換えができないことの 2 点がデメリットとなる．そこで，リーチをかけられる状態でもリーチをしないという戦略がある．ダマテンは，リーチしなくても跳満以上になる場合や，聴牌したのが局が始まってすぐで，手を高くできそうだったり良形になりそうだったりする場合[2][3]などに用いられる戦略である．本研究でも[2][3]の聴牌すればリーチし，他家がリーチしたときや一定の順目になったときに一向聴以下ならオリる戦略をとる．

2.4. 牌効率を重視する戦略

1.2 節で述べた通り, 麻雀 AI で有望とされるのが, 手牌および相手の捨牌から各牌の残り枚数を求め, 最も和了率が高くなるように捨牌を選ぶ手法である. この手法は平均的にはそれなりの和了率が得られる一方, 放銃を考えていないため高得点を振り込んでしまうことも多い． 牌効率重視の ComAI1 クラスでは自摸牌でシャンテン数が進んだ場合，シャンテン数が減らないように牌を切っていき，シャンテン数が減らない牌が 2 つ以上ある場合，字牌, 1・9 牌, 2・8 牌, 3・7 牌, 4・ 6 牌, 5 牌と切っていく

2.5. 放銃を避ける戦略

本研究で使用した最下位を避ける AI として comAI2 クラスを作成した．普段は牌効率重視で打牌していくが，オリる際は捨牌を現物，単騎自牌，スジ 19，単騎以外の字牌， スジ 28・スジ 37，ムスジ 19・カタスジ 456,・ムスジ 28・ ムスジ 37，ムスジ 456 の順に落としていく[2][3]．本研究で様々な条件をつけてそのオリる判断をし，一度オリたらもう一度聴牌を目指したり，回し打ちはしない戦略をとる．

3. 麻雀プログラム

本章では, 本研究で作成した麻雀プログラムの各クラスについて説明する．

本研究では, Python を用いて麻雀のプログラムを作成した．付録に本研究で作成したプログラムのソースコードを示す．翻数， 府数，役判定，点数計算には Python の「mahjong」というライブラリを使用した[6]．本研究で作成した麻雀プログラムは, CPU と対戦，及び対人戦を行えるようになっている．

図 1 に本研究で作成したプログラムの実行の様子を示す．

以下に本研究で作成した麻雀プログラムの各クラスについて説明する．

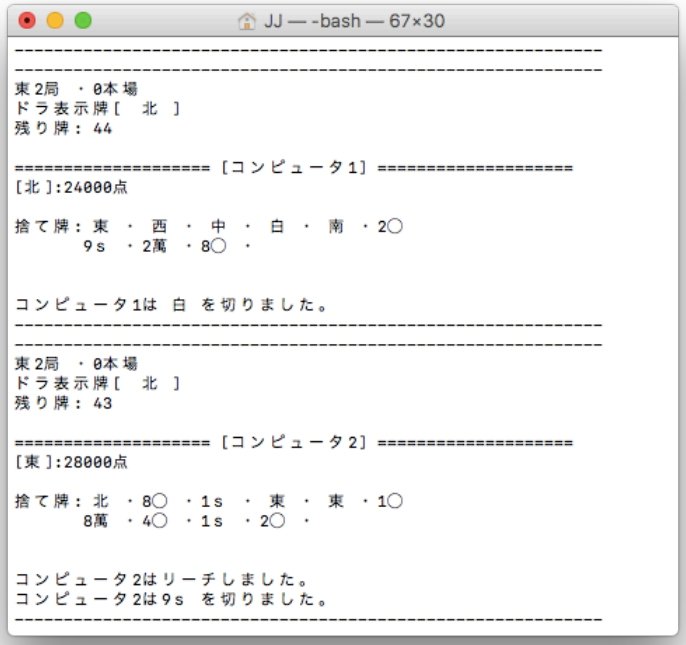

図 1 作成した麻雀プログラムの実行の様子

3.1. Agari_search クラス

Agari_search クラスは，役判定と点数計算をするクラスである．図 1 に Agari_search クラスのクラス図を示す．

Agari_search	役判定と点数計算をするクラス
caluculator: HandCalculator honors: str man: str pin: str sou: str result: int	計算用クラスのインスタンスを生成 役牌 萬子 筒子 索子 計算した点数
agari(tumohai, dora, is_tsumo, is_riichi, is_ippatsu, is_rinshan, is_chankan, is_houtei, is_daburu_riichi, is_nagashi_mangan, is_tenhou, is_renhou, is_chiihou, player_wind, round_wind, ankanhai, minkanhai, ponhai, chiihai) print_hand_result() shanten()	役判定と点数計算 点数計算した結果の表示 シャンテン数の計算

tehai_henkan(tehai)	手牌を計算できるように変換

図 2　Agari_search クラスのクラス図

3.2. ComAI1 クラス

ComAI1 クラスは, 牌効率のみのコンピュータのクラスである. 図 2 に ComAI1 クラスのクラス図を示す.

ComAI1	牌効率のみのコンピュータのクラス
ankan()	暗カン
chii(taku, jansi, sutehai, mae_ags, ato_ags)	チー
chii2(jansi, chii_value, ato_ags, ags)	2 種類以上チーできる場合のチー
dahai(taku, jansi, shantensuu, teki)	打牌
kakan()	加カン
minkan()	明カン
pon(taku, jansi, sutehai, mae_ags, ato_ags)	ポン
riichi(taku, jansi, teki)	リーチ
ron()	ロン

図 3　ComAI1 クラスのクラス図

3.3. ComAI2 クラス

ComAI2 クラスは, 4 位を取らない AI のクラスである. 図 3 に ComAI2 クラスのクラス図を示す.

ComAI2	4 位を取らない AI クラス
ankan()	暗カン
chii(taku, jansi, sutehai, mae_ags, ato_ags)	チー
chii2(jansi, chii_value, ato_ags, ags)	2 種類以上チーできる場合のチー
dahai(taku, jansi, shantensuu, teki)	打牌
kakan()	加カン
keikai(teki, taku, jansi, junni)	警戒する敵を決める
minkan()	明カン
ori(taku, jansi, shantensuu, teki, keikai)	オリ
ori_judge(teki, taku, keikai, jansi, shantensuu, junni)	オリるかどうかの判断
pon(taku, jansi, sutehai, mae_ags, ato_ags)	ポン
riichi(taku, jansi, teki)	リーチ
ron()	ロン

図 4　ComAI2 クラスのクラス図

3.4. Computer クラス

Computer クラスは，コンピュータの挙動を表すクラスである．図 4 に Computer クラスのクラス図を示す.

Computer	コンピュータの挙動を表すクラス
agari_flag: bool	上がりのフラグ
ags: Agari_search	和了時の役判定
anzenpai: list	安全牌
comAI: ComAI	AI
huriten2: bool	フリテンかどうか
jansi: Jansi	雀士
move_tenbou: list	動かす点棒
naki: bool	鳴いてるかどうか
name: str	名前
taku: Taku	卓
teki: list	敵のリスト
tenapi_flag: bool	テンパイのフラグ

add_tenbou(tenbou)	テンパイ時に点棒を加える
add_tenbou2()	和了時に点棒を加える
agari_kan()	上がるかカンする
append_hai(hai)	牌を加える
calculation()	点数を計算する
change_player_wind()	自風を変える
chii(sutehai)	チー
dahai()	打牌
do_riichi()	リーチするかどうか
finish_round(dareka_agari)	１局終わる時の挙動
game()	コンピュータが麻雀する時の挙動
get_agari_flag()	上がりフラグのゲッター
get_is_riichi()	リーチしてるかどうかのゲッター
get_move_tenbou()	動かす点棒のゲッター
get_player_wind()	自風のゲッター
get_sutehai()	捨牌のゲッター
get_tenbou()	点棒のゲッター
get_tenpai_flag()	テンパイフラグのゲッター
haipai()	配牌する
kakan()	加カン
minkan(sutehai)	明カン
pon(sutehai)	ポン
reset_agari_flag()	上がりフラグをリセットする
riipai()	理牌
ron(sutehai, teki_name)	ロン
search_agarihai()	上がり牌の探索
set_anzenpai(anzenpai)	安全牌のセッター
set_player_wind(player_wind)	自風のセッター
set_teki(teki)	敵のセッター
show_name()	名前の表示
show_player_wind()	自風の表示
show_tehai()	手牌の表示
show_tenbou()	点棒の表示
show_tumo()	ツモの表示
sub_tenbou(tenbou2)	点棒を減らす
unit_nakihai()	カンとポンとチーを鳴き牌に加える

図 5　Computer クラスのクラス図

3.5. Human クラス

Human クラスは，人間の挙動を表すクラスである．図 5 に Human クラスのクラス図を示す.

Human	人間の挙動を表すクラス
agari_flag: bool	上がりフラグ
ags: Agari_search	和了時の役判定
huriten2: bool	フリテンしてるかどうか
jansi: Jansi	雀士
move_tenbou: list	動かす点棒
naki: bool	鳴いてるかどうか
name: str	名前
taku: Taku	卓
tenapi_flag: bool	テンパイフラグ

add_tenbou(tenbou)	テンパイ時に点棒を加える
add_tenbou2()	和了時に点棒を加える
agari_kan()	上がるかカンする
append_hai(hai)	牌を加える
calculation()	点数を計算する
change_player_wind()	自風を変える
chii(sutehai)	チー
dahai()	打牌
do_riichi()	リーチするかどうか
finish_round(dareka_agari)	１局終わる時の挙動
game()	コンピュータが麻雀する時の挙動
get_agari_flag()	上がりフラグのゲッター
get_is_riichi()	リーチしてるかどうかのゲッター
get_move_tenbou()	動かす点棒のゲッター
get_player_wind()	自風のゲッター
get_sutehai()	捨て牌のゲッター
get_tenbou()	点棒のゲッター
get_tenpai_flag()	テンパイフラグのゲッター
haipai()	配牌する
kakan()	加カン
minkan(sutehai)	明カン
pon(sutehai)	ポン
reset_agari_flag()	上がりフラグをリセットする
riipai()	理牌
ron(sutehai, teki_name)	ロン
search_agarihai()	上がり牌の探索
set_anzenpai(anzenpai)	安全牌のセッター
set_player_wind(player_wind)	自風のセッター
set_teki(teki)	敵のセッター
show_name()	名前の表示
show_player_wind()	自風の表示
show_tehai()	手牌の表示
show_tenbou()	点棒の表示
show_tumo()	ツモの表示
sub_tenbou(tenbou2)	点棒を減らす
unit_nakihai()	カンとポンとチーを鳴き牌に加える

図 6　Human クラスのクラス図

3.6. Jansi クラス

Jansi クラスは，雀士を表すクラスである．図 6 に Jansi クラスのクラス図を示す．

Jansi	雀士を表すクラス
anknahai: list	暗カン牌
chankanhai: int	加カン牌
chiihai: list	チー牌
dahai_count: int	打牌した数
is_chankan: bool	カカンしたかどうか
is_chiihou: bool	地和かどうか
is_daburu_riichi: bool	ダブルリーチかどうか
is_haitei: bool	ハイテイかどうか
is_ippatsu: bool	一発かどうか
is_nagashi_mangan: bool	流し満貫かどうか
is_renhou: bool	レンホーかどうか
is_tenhou: bool	テンホーかどうか
is_tsumo: bool	ツモかどうか
kanji: Kanji	見やすいように漢字に変える
minkanhai: list	明カン牌
player_wind: int	自風
ponhai: list	ポン牌
riichi_dahai_count: int	リーチ後の打牌した数
riichi_hai: int	リーチした牌
rinshanhai: int	リンシャン牌
sutehai: list	捨牌
tehai: list	手牌
tenbou: int	点棒
tumohai: int	ツモ牌
unit_tehai: list	手牌と鳴いた牌を合わせる

add_tenbou(tensuu)	点棒を加える
ankan(value, taku)	暗カン
change_player_wind()	自風を変える
chii(value)	チー
dahai(value)	打牌
get_ankanhai()	暗カン牌のゲッター
get_chiihai()	チー牌のゲッター
get_is_chankan()	チャンカンのゲッター
get_is_chiihou()	地和のゲッター
get_is_daburu_riichi()	ダブルリーチのゲッター
get_is_haitei()	ハイテイのゲッター
get_is_houtei()	ホウテイのゲッター
get_is_ippatsu()	一発のゲッター
get_is_nagashi_mangan()	流し満貫のゲッター
get_is_renhou()	人和のゲッター
get_is_riichi()	リーチのゲッター
get_is_rinshan()	リンシャンのゲッター
get_is_tenhou()	天和のゲッター
get_is_tsumo()	ツモのゲッター
get_minkanhai()	明カン牌のゲッター
get_player_wind()	自風のゲッター
get_ponhai()	ポン牌のゲッター
get_sutehai()	捨牌のゲッター
get_tehai()	手牌のゲッター
get_tumohai()	ツモ牌のゲッター
get_unit_tehai()	手牌と鳴き牌のゲッター
haipai(yama)	配牌
kakan(value, taku)	加カン
naki()	鳴いてる牌
pon(value)	ポン
reset_hai()	捨牌、手牌、鳴き牌のリセット
reset_yaku()	役のリセット
riipai()	理牌
set_is_chankan()	チャンカンのセッター
set_is_chiihou()	地和のセッター
set_is_daburu_riichi()	ダブルリーチのセッター
set_is_haitei()	ハイテイのセッター
set_is_houtei()	ホウテイのセッター

set_is_ippatsu()	一発のセッター
set_is_nagashi_mangan()	流し満貫のセッター
set_is_renhou()	人和のセッター
set_is_riichi()	リーチのセッター
set_is_rinshan(rinshan)	リンシャンのセッター
set_is_tenhou()	天和のセッター
set_is_tsumo(tsumo)	ツモのセッター
set_minkanhai()	明カン牌のセッター
set_player_wind(player_wind)	自風のセッター
set_ponhai()	ポン牌のセッター
set_sutehai()	捨て牌のセッター
set_tehai(tehai)	手牌のセッター
set_tumohai(tumohai)	ツモ牌のセッター
show_nakihai()	鳴き牌の表示
show_sutehai()	捨牌の表示
show_tenbou()	点棒の表示
tumo(yama)	山から牌を加える
unit_agari()	和了のために手牌と鳴き牌を合わせる
unit_nakihai()	ポンとチーとカンを合わせる

図 7　Jansi クラスのクラス図

3.7. Kanji クラス

Kanji クラスは, 牌を漢字にして表示するクラスである. 図 7 に Kanji クラスのクラス図を示す.

Kanji	牌を漢字にして表示するクラス
get_kanji(value)	漢字にして表示

図 8　Kanji クラスのクラス図

3.8. Taku クラス

Taku クラスは, 卓を表すクラスである. 図 8 に Taku クラスのクラス図を示す.

Taku	卓を表すクラス
dora: list	ドラ
honba: int	本場数
jansi1_anzenpai: list	雀士 1 の安全牌
jansi1_nakihai: list	雀士 1 の鳴き牌
jansi1_riichi: bool	雀士 1 がリーチしてるかどうか
jansi1_sutehai: list	雀士 1 の捨牌
jansi2_anzenpai: list	雀士 2 の安全牌
jansi2_nakihai: list	雀士 2 の鳴き牌
jansi2_riichi: bool	雀士 2 がリーチしてるかどうか
jansi2_sutehai: list	雀士 2 の捨牌
jansi3_anzenpai: list	雀士 3 の安全牌
jansi3_nakihai: list	雀士 3 の鳴き牌
jansi3_riichi: bool	雀士 3 がリーチしてるかどうか
jansi3_sutehai: list	雀士 3 の捨牌
jansi4_anzenpai: list	雀士 4 の安全牌
jansi4_nakihai: list	雀士 4 の鳴き牌
jansi4_riichi: bool	雀士 4 がリーチしてるかどうか
jansi4_sutehai: list	雀士 4 の捨牌
kan: int	カンした数
oki_riibou: int	卓上にあるリーチ棒
round: int	局数
round_wind: int	場風
start_East: int	最初の東場の雀士
yama: list	山

add_dora()	ドラを増やす
add_honba()	本場数を増やす
add_kan()	カンした数を増やす
add_round()	局数を増やす
add_uradora()	裏ドラを増やす
change_round_wind()	場風を変える
get_dora()	ドラのゲッター
get_honba()	本場のゲッター
get_oki_riibou()	置いてあるリー棒のゲッター
get_rinshan()	リンシャン牌のゲッター
get_round()	局数のゲッター
get_round_wind()	場風のゲッター
get_yama()	山のゲッター
reset_dora()	ドラをリセットする
reset_honba()	本場数をリセットする
reset_round()	局数をリセットする
set_oki_riibou()	置いてあるリー棒をセットする
set_round(round)	局数をセットする
set_start_East(start_East)	始めの東場をセットする
show_dora()	ドラを表示
show_honba()	本場数を表示
show_round()	局数を表示
show_yama()	山を表示
yama_reset()	山をリセットする

図 9　Taku クラスのクラス図

3.9. Yonin クラス

Yonin クラスは, 四人麻雀を表すクラスである. 図 9 に Yonin クラスのクラス図を示す.

Yonin	四人麻雀を表すクラス
change_jansi(jansi1, jansi2, jansi3, jansi4, now_jansi)	牌を捨てる雀士を変える
change_teki(jansi1, jansi2, jansi3, jansi4, now_jansi)	敵の雀士を変える
finish(taku, jansi1, jansi2, jansi3, jansi4, is_ron)	終局時の処理
nakihai(taku, jansi1, jansi2, jansi3, jansi4)	鳴き牌を卓上に置く
play_game()	ゲームをプレイする
result_round(jansi1, jansi2,jansi3	終局時の結果

図 10　Yonin クラスのクラス図

4. 研究内容

本研究では, Python で自作した牌効率を重視した麻雀ゲームのプログラム(以下牌効率重視 AI と呼ぶ)を元に, 最下位を避けることを重視した麻雀 AI を作成する. 作成した麻雀 AI を牌効率重視 AI と対戦させて最下位率を調査し, どのような場合に最下位率を下げられているかを検証する.

本研究では 1) 一定枚数の捨牌をするまでにテンパイできなければオリる 2) 他家がリーチした場合に向聴数によってオリる 3) 他家がリーチした場合に 4 位との点差によってオリる 4) 4 位がリーチした場合にオリる　の 4 つの条件の麻雀 AI を作成し, オリる条件となる手牌の枚数を変えて牌効率重視 AI と対戦させ, 最も最下位率が低くなる枚数を求める.

5. 実験結果・統計的検証および考察

5.1. 実験結果

以下の表に 300 回対局したデータを示す. 最下位率は 300 回中の最下位をとった確率を表している.

表 1　捨牌枚数と最下位率の関係

捨牌枚数	8	9	10	11	12	13	14	15	16	オリない
最下位率	0.27	0.23	0.26	0.24	0.29	0.28	0.28	0.28	0.22	0.26

表 2　他家がリーチした場合の向聴数によるオリる条件と最下位率の関係

	一向聴ならオリない	二向聴ならオリない	自家が 4 位 + 一向聴ならオリない	自家が 4 位 + 二向聴ならオリない	常にオリる

最下位率	0.23	0.25	0.24	0.29	0.26

表 3 他家がリーチした場合に 4 位との点差によるオリる条件と最下位率の関係

	点差が 1000 点ならオリない	点差が 2000 点ならオリない	点差が 3900 点ならオリない	点差が 7700 点ならオリない
最下位率	0.28	0.26	0.29	0.28

表 4　4 位がリーチした場合のオリる条件と最下位率の関係

	4 位がリーチした場合のみオリる	自家が親で 4 位がリーチした場合のみオリる
最下位率	0.03	0.12

5.2. 統計的検証および考察

4 節で最下位を避ける条件について有効であったかを考察する.

1) 一定枚数の捨牌をするまでにテンパイできなければオリる

16 順目の捨牌の時にオリた場合に一番最下位率を下げられることが示されたが, 捨牌の枚数によって多少の差はあるものの最下位率はあまり下がらないことが示された.

2) 他家がリーチした場合に向聴数によってオリる

二向聴の場合は最下位率があまり下がらないが, 一向聴からなら少しだけ最下位率が下がることが示された.

3) 他家がリーチした場合に 4 位との点差によってオリる

他家がリーチした場合に 4 位との点差がどれだけあっても, どれも最下位率が上がってしまうことが示された.

4) 4 位がリーチした場合にオリる

最下位率が大幅に下がることが示された. 特に「4 位がリーチした場合のみオリる」条件は 300 回中 8 回しか最下位を取らなかったためとても効果があったと思う.

以下ではそれぞれの結果について統計的に検証する.

4 人プレイであるので, 各 AI の戦略に優劣が無ければ, 1 位~4 位になる確率はそれぞれ 25%である. そこで, 最下位率が 25%と仮定した時に, 統計上有意な差があるかを検証する.

勝率 p の勝負を N 回行った場合、標準偏差 s は以下の式で表される.

$$s = \sqrt{N * p * (1 - p)}$$

p=0.25 と仮定すると、N=300 ならば標準偏差は

$$\sqrt{300 * 0.25 * 0.75} = 7.50$$

となる. 信頼区間 95% となるのは最下位回数が平均値からの差が 7.50*1.96=14.7 となる区間である. したがって, 最下位率が 25% ならば, 300 試合すれば 95%の確率で最下位回数は 75±14.7 , , 最下位率は 25±4.9%に収まる.

以上のことから, 1) 一定枚数の捨牌をするまでにテンパイできなければオリる, 2) 他家がリーチした場合に向聴数によってオリる, 3) 他家がリーチした場合に 4 位との点差によってオリる　の 3 つの条件は最下位率が 20%～30%の範囲に収まっており誤差である可能性が高いが, 4) 4 位がリーチした場合にオリる は統計的に有意であると考えられる. したがって, 4 位がリーチした場合はオリるべきであると言える.

また, [2][3]では，他家がリーチした場合，自分が聴牌しているなら勝負し，一向聴以下ならオリるべきとされているが，表 2 からはオリても最下位率は変わらない. これは相手が牌効率のみのコンピュータであったため， 人間相手とは異なる結果が得られたと言える.

6. 結論・今後の課題

本研究では, Python で最下位を避けることを重視した AI を作成した. しかし, 麻雀は不確定非完全情報ゲームであり, ランダム要素が絡むのでどの局面において必ずしも正しい結果となっているとは言えない.

今後の課題とし, 様々な対戦相手と対戦しても同じように最下位を取らないような AI を作成していきたいと考えている.

参考文献

[1] M.LEAGUE(M リーグ) 公式サイト, https://m-league.jp

[2] とつげき東北. *科学する麻雀*. 講談社. (2004).

[3] とつげき東北. *おしえて！科学する麻雀*. 洋泉社. (2009).

[4] 麻雀 AI Microsoft Suphx が人間のトッププレイヤーに匹敵する成績を達成, Japan News Center, Mictosoft (2019/8/29) https://news.microsoft.com/ja-jp/2019/08/29/190829-mahjong-ai-microsoft-suphx/

[5] とつげき東北, ASAPIN, 水上直紀, マイクロソフト「麻雀 AI」の衝撃…麻雀界はここまで激変する, 現代ビジネス, 講談社 (2019/9/30) https://gendai.ismedia.jp/articles/-/67507

[6]Python ライブラリの「麻雀」(mahjong)って？, https://qiita.com/FJyusk56/items/8189bcca3849532d095f

付録

本研究で作成した麻雀のプログラムのソースを以下に示す.

<Agari_search.py>

```
# -*- coding: utf-8 -*-
from Taku import Taku
from Kanji import Kanji
from Jansi import Jansi
from mahjong.shanten import Shanten
#計算
from mahjong.hand_calculating.hand import HandCalculator
#麻雀牌
from mahjong.tile import TilesConverter
#役，オプションルール
from mahjong.hand_calculating.hand_config import HandConfig, OptionalRules
#鳴き
from mahjong.meld import Meld
#風(場&自)
from mahjong.constants import EAST, SOUTH, WEST, NORTH
import copy

#役判定と点数を計算する
class Agari_search(object):

    def __init__(self):
        #HandCalculator(計算用クラス)のインスタンスを生成
        self.calculator = HandCalculator()
        self.man = ''
        self.pin = ''
        self.sou = ''
        self.honors = ''
        self.result = None

    #結果出力用
    def print_hand_result(self):
         print("")
         #翻数，符数
         print(str(self.result.han) + "飜",end='')
         print("(" + str(self.result.fu) + "符):",end='')
         #役
```

```
        print(str(self.result.yaku))
        #点数(ツモアガリの場合[左：親失点, 右:子失点], ロンアガリの場合[左:放銃者失点, 右:0])
        print("=" *4,end='')
        print(str(self.result.cost['main']) + "点,",end='')
        print(str(self.result.cost['additional']) + "点",end='')
        print("=" *4)
        #符数の詳細
        print('')

#シャンテン数の計算
def shanten(self):
    #Shanten(シャンテン数計算用クラス)のインスタンスを生成
    shanten = Shanten()

    #手牌 14 枚
    tiles = TilesConverter.string_to_34_array(self.sou, self.man, self.pin, self.honors)

    #計算
    result = shanten.calculate_shanten(tiles)
    return result

#アガリの表示
def agari(self, tumohai, dora, is_tsumo, is_riichi,
                is_ippatsu, is_rinshan, is_chankan, is_haitei,
                is_houtei, is_daburu_riichi, is_nagashi_mangan,
                is_tenhou, is_renhou, is_chiihou, player_wind, round_wind,
                ankanhai, minkanhai, ponhai, chiihai):

    #アガリ形(honors=1:東, 2:南, 3:西, 4:北, 5:白, 6:發, 7:中)
    tiles = TilesConverter.string_to_136_array(self.sou, self.man, self.pin, self.honors)

    #アガリ牌
    tumohai2 = ''
    if tumohai < 9:
        tumohai2 = str(tumohai + 1)
        win_tile = TilesConverter.string_to_136_array(pin=tumohai2)[0]
    elif tumohai >= 9 and tumohai < 18:
        tumohai2 = str(tumohai - 8)
        win_tile = TilesConverter.string_to_136_array(man=tumohai2)[0]
```

```
elif tumohai >= 18 and tumohai < 27:
    tumohai2 = str(tumohai - 17)
    win_tile = TilesConverter.string_to_136_array(sou=tumohai2)[0]
else:
    tumohai2 = str(tumohai - 26)
    win_tile = TilesConverter.string_to_136_array(honors=tumohai2)[0]

#鳴き(チー:CHI, ポン:PON, カン:KAN(True:ミンカン,False:アンカン), カカン:CHANKAN)
melds = []
#暗カン
if len(ankanhai) > 0:
    for i in range(len(ankanhai)):
        ankanhai2 = ''
        if int(ankanhai[i][0]) < 9:
            for j in range(4):
                ankanhai2 += str(int(ankanhai[i][j]) + 1)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(pin=ankanhai2), False))
        elif int(ankanhai[i][0]) >= 9 and int(ankanhai[i][0]) < 18:
            for j in range(4):
                ankanhai2 += str(int(ankanhai[i][j]) - 8)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(man=ankanhai2), False))
        elif int(ankanhai[i][0]) >= 18 and int(ankanhai[i][0]) < 27:
            for j in range(4):
                ankanhai2 += str(int(ankanhai[i][j]) - 17)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(sou=ankanhai2), False))
        else:
            for j in range(4):
                ankanhai2 += str(int(ankanhai[i][j]) - 26)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(honors=ankanhai2), False))
#明カン
if len(minkanhai) > 0:
    for i in range(len(minkanhai)):
        minkanhai2 = ''
        if int(minkanhai[i][0]) < 9:
            for j in range(4):
```

```
                minkanhai2 += str(int(minkanhai[i][j]) + 1)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(pin=minkanhai2), True))
        elif int(minkanhai[i][0]) >= 9 and int(minkanhai[i][0]) < 18:
            for j in range(4):
                minkanhai2 += str(int(minkanhai[i][j]) - 8)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(man=minkanhai2), True))
        elif int(minkanhai[i][0]) >= 18 and int(minkanhai[i][0]) < 27:
            for j in range(4):
                minkanhai2 += str(int(minkanhai[i][j]) - 17)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(sou=minkanhai2), True))
        else:
            for j in range(4):
                minkanhai2 += str(int(minkanhai[i][j]) - 26)
            melds.append(Meld(Meld.KAN,
                TilesConverter.string_to_136_array(honors=minkanhai2), True))
#ポン
if len(ponhai) > 0:
    for i in range(len(ponhai)):
        ponhai2 = ''
        if int(ponhai[i][0]) < 9:
            for j in range(3):
                ponhai2 += str(int(ponhai[i][j]) + 1)
            melds.append(Meld(Meld.PON,
                TilesConverter.string_to_136_array(pin=ponhai2)))
        elif int(ponhai[i][0]) >= 9 and int(ponhai[i][0]) < 18:
            for j in range(3):
                ponhai2 += str(int(ponhai[i][j]) - 8)
            melds.append(Meld(Meld.PON,
                TilesConverter.string_to_136_array(man=ponhai2)))
        elif int(ponhai[i][0]) >= 18 and int(ponhai[i][0]) < 27:
            for j in range(3):
                ponhai2 += str(int(ponhai[i][j]) - 17)
            melds.append(Meld(Meld.PON,
                TilesConverter.string_to_136_array(sou=ponhai2)))
        else:
            for j in range(3):
```

```
                ponhai2 += str(int(ponhai[i][j]) - 26)
            melds.append(Meld(Meld.PON,
                TilesConverter.string_to_136_array(honors=ponhai2)))
#チー
if len(chiihai) > 0:
    for i in range(len(chiihai)):
        chiihai2 = ''
        if int(chiihai[i][0]) < 9:
            for j in range(3):
                chiihai2 += str(int(chiihai[i][j]) + 1)
            melds.append(Meld(Meld.CHI,
                TilesConverter.string_to_136_array(pin=chiihai2)))
        elif int(chiihai[i][0]) >= 9 and int(chiihai[i][0]) < 18:
            for j in range(3):
                chiihai2 += str(int(chiihai[i][j]) - 8)
            melds.append(Meld(Meld.CHI,
                TilesConverter.string_to_136_array(man=chiihai2)))
        elif int(chiihai[i][0]) >= 18 and int(chiihai[i][0]) < 27:
            for j in range(3):
                chiihai2 += str(int(chiihai[i][j]) - 17)
            melds.append(Meld(Meld.CHI,
                TilesConverter.string_to_136_array(sou=chiihai2)))
        else:
            for j in range(3):
                chiihai2 += str(int(chiihai[i][j]) - 26)
            melds.append(Meld(Meld.CHI,
                TilesConverter.string_to_136_array(honors=chiihai2)))

#ドラ
dora_indicators = []
dora2 = ''
for i in dora:
    if i < 9:
        dora2 = str(i + 1)
        dora_indicators.append(TilesConverter.string_to_136_array(pin=dora2)[0])
    elif i >= 9 and i < 18:
        dora2 = str(i - 8)
        dora_indicators.append(TilesConverter.string_to_136_array(man=dora2)[0])
    elif i >= 18 and i < 27:
```

```
                dora2 = str(i - 17)
                dora_indicators.append(TilesConverter.string_to_136_array(sou=dora2)[0])
            else:
                dora2 = str(i - 26)
                dora_indicators.append(TilesConverter.string_to_136_array(honors=dora2)[0])

        #オプション
        config = HandConfig(is_tsumo, is_riichi, is_ippatsu,
                            is_rinshan, is_chankan,is_haitei, is_houtei,
                            is_daburu_riichi, is_nagashi_mangan, is_tenhou,
                            is_renhou, is_chiihou, player_wind, round_wind,
                            options=OptionalRules
                                (has_open_tanyao=True, has_aka_dora=False,
                                 has_double_yakuman=True,
                                 kazoe_limit = HandConfig.KAZOE_LIMITED))



        #計算
        self.result = self.calculator.estimate_hand_value(tiles, win_tile,melds,dora_indicators, config)
        #受け取る点棒の値
        tenbou = [self.result.cost['main'] , self.result.cost['additional'] ]
        return tenbou

#手牌をシャンテン数が計算できるように変換する
def tehai_henkan(self, tehai):
        self.man = ''
        self.pin = ''
        self.sou = ''
        self.honors = ''
        for i in tehai:
            if i < 9:
                self.man += str(i+1)
            elif i >= 9 and i < 18:
                self.pin += str(i-8)
            elif i >= 18 and i < 27:
                self.sou += str(i-17)
            elif i >= 27 :
                self.honors+= str(i-26)
```

<ComAI1.py>

```
# -*- coding: utf-8 -*-
from Agari_search import Agari_search
from Taku import Taku
from mahjong.constants import EAST, SOUTH, WEST, NORTH
import copy

#牌効率のみのコンピュータの挙動
class ComAI1(object):

    #打牌するときの AI
    def dahai(self, taku, jansi, shantensuu, teki):
        #シャンテン数が減らない牌の検索
        ags1 = Agari_search()
        haikouritu = []
        #シャンテン数が進む牌の検索
        ags2 = Agari_search()
        #順に切るための関数
        value = -1
        while True:
            #仮の手牌を作る
            tehai1 = copy.deepcopy(jansi.tehai)
            value += 1
            #手牌全て切り終わったらループを抜ける
            if value == len(tehai1):
                break
            #切る牌を仮に入れて置く
            kiruhai = copy.deepcopy(tehai1[value])
            #左から順に一枚ずつ仮に切っていく
            del tehai1[value]
            ags1.tehai_henkan(tehai1)
            #シャンテン数が増えたならもう一度最初からやる
            if ags1.shanten() > shantensuu:
                continue
            #シャンテン数が変わらない場合(tehai1 は 13 枚)
            else:
                #13 枚の時、仮に一枚入れてシャンテン数が進んだ牌を入れる
                susumu_list = []
                #シャンテン数が進む牌を探す
```

```
            for i in range(34):
                #仮りの手牌 2
                tehai2 = copy.deepcopy(tehai1)
                #仮りの手牌 2 に一つ牌を入れてみる
                tehai2.append(i)
                ags2.tehai_henkan(tehai2)
                #シャンテン数が進んだならリストに入れる
                if ags2.shanten() < ags1.shanten():
                    susumu_list.append(i)
            #手牌が進む残り牌の数を数えるための関数
            matihaisuu = int(len(susumu_list)) * 4
            #卓上で公になってる牌(自分の手牌,捨て牌,鳴いてる牌、ドラ)
            koukaihai = [jansi.tehai, taku.dora,
                         taku.jansi1_sutehai,
                         taku.jansi2_sutehai,
                         taku.jansi3_sutehai,
                         taku.jansi4_sutehai,
                         taku.jansi1_nakihai,
                         taku.jansi2_nakihai,
                         taku.jansi3_nakihai,
                         taku.jansi4_nakihai,]
            #手牌が進む残り牌の数を数える(減らしていく)
            for k in koukaihai:
                for i in k:
                    for j in susumu_list:
                        if i == j:
                            matihaisuu -= 1
            #haikouritu に[切った牌, 出た値]を入れる
            haikouritu.append([kiruhai ,matihaisuu])
    #テンパイなら積もった牌を捨てる
    if len(haikouritu) > 1:
        #待ち牌が多くなるものを選ぶ
        value2 = 0
        haikouritu2 = []
        for i in haikouritu:
            if i[1] > value2:
                haikouritu2 = [i[0]]
                value2 = i[1]
            elif i[1] == value2:
```

```
                haikouritu2.append(i[0])
        #東南西北→白發中→１・９牌→２・８牌→３・７牌→４・６牌→５牌と切っていく
        haikouritu3 = -1
        #if 文の階層を残して置くための関数
        count = 10
        if len(haikouritu2) > 0:
            for i in haikouritu2:
                if i == 27 or i == 28 or i == 29 or i == 30 and count > 0:
                    haikouritu3 = i
                    count = 1
                elif (i == 31 or i == 32 or i == 33) and count > 1:
                    haikouritu3 = i
                    count = 2
                elif (i == 0 or i == 8 or i == 9 or¥
                        i == 17 or i == 18 or i == 26) and count > 2:
                    haikouritu3 = i
                    count = 3
                elif (i == 1 or i == 7 or i == 10 or¥
                        i == 16 or i == 19 or i == 25) and count > 3:
                    haikouritu3 = i
                    count = 4
                elif (i == 2 or i == 6 or i == 11 or¥
                        i == 15 or i == 20 or i == 24) and count > 4:
                    haikouritu3 = i
                    count = 5
                elif (i == 3 or i == 5 or i == 12 or¥
                        i == 14 or i == 21 or i == 23) and count > 5:
                    haikouritu3 = i
                    count = 6
                elif count > 6:
                    haikouritu3 = i
                    count = 7
        #手牌から value に当てはまる牌の位置を覚える配列
        index = jansi.tehai.index(haikouritu3)
    else:
        #一番後ろを返す
        index = len(jansi.tehai) - 1
    #打牌する牌の場所を返す
    return index
```

```
#リーチするかどうかの AI
def riichi(self, taku, jansi, teki):
    riichi = 'y'
    return riichi

#ロンするかどうかの AI
def ron(self):
    return 'y'

#暗カンするときの AI
def ankan(self):
    return 'n'

#明カンするときの AI
def minkan(self):
    return 'n'

#加カンするときの AI
def kakan(self):
    return 'y'

#ポンするときの AI
def pon(self, taku, jansi, sutehai, mae_ags, ato_ags):
    ponsuru = 'n'
    #捨て牌が役牌ならポンする
    if sutehai == 31 or sutehai == 32 or sutehai == 33 or¥
        (jansi.player_wind == EAST and sutehai == 27)or¥
        (jansi.player_wind == SOUTH and sutehai == 28)or¥
        (jansi.player_wind == WEST and sutehai == 29)or¥
        (jansi.player_wind == NORTH and sutehai == 30)or¥
        (taku.round_wind == EAST and sutehai == 27)or¥
        (taku.round_wind == SOUTH and sutehai == 28)or¥
        (taku.round_wind == WEST and sutehai == 29)or¥
        (taku.round_wind == NORTH and sutehai == 30):
        #シャンテン数が進んだらポンする(元から３枚あるならポンしない)
        if mae_ags > ato_ags:
            ponsuru = 'y'
    #すでに鳴いているならポンする
```

```
        elif jansi.naki() == True:
            #シャンテン数が進んだらポンする(元から３枚あるならポンしない)
            if mae_ags > ato_ags:
                ponsuru = 'y'
        return ponsuru

    #チーするときの AI
    def chii(self, taku, jansi, sutehai, mae_ags, ato_ags):
        chiisuru = 'n'
        #すでに鳴いているなら鳴く
        if jansi.naki() == True:
            #シャンテン数が進んだらチーする
            if mae_ags > ato_ags:
                chiisuru = 'y'
        return chiisuru

    #２パターン以上チーできる場合
    def chii2(self, jansi, chii_value, ato_ags, ags):
        value = -1
        for i in chii_value:
            value += 1
            jansi.chii(i)
            ags.tehai_henkan(jansi.tehai)
            jansi.tehai.append(i[0])
            jansi.tehai.append(i[1])
            jansi.tehai.append(i[2])
            del jansi.chiihai[-1]
            if ato_ags == ags.shanten():
                break
        return value
```

<ComAI2.py>

```
# -*- coding: utf-8 -*-
from Agari_search import Agari_search
from Taku import Taku
import copy
import collections
from mahjong.constants import EAST, SOUTH, WEST, NORTH
```

```
#４位を取らない AI の挙動
class ComAI2(object):

    #打牌するときの AI
    def dahai(self, taku, jansi, shantensuu, teki):
        #自分の順位と最下位を入れる
        judge_junni = self.search_junni(taku, jansi, teki)
        #誰かがリーチしてたり、残りの山の数で降り始めるための関数
        keikai = self.keikai(teki, taku, jansi, judge_junni)
        orihai = self.ori(taku, jansi, shantensuu, teki, keikai)
        if self.ori_judge(teki, taku, keikai, jansi, shantensuu, judge_junni) == False¥
            or orihai == -1:
            #シャンテン数が減らない牌の検索
            ags1 = Agari_search()
            haikouritu = []
            #シャンテン数が進む牌の検索
            ags2 = Agari_search()
            #順に切るための関数
            value = -1
            while True:
                #仮の手牌を作る
                tehai1 = copy.deepcopy(jansi.tehai)
                value += 1
                #手牌全て切り終わったらループを抜ける
                if value == len(tehai1):
                    break
                #切る牌を仮に入れて置く
                kiruhai = copy.deepcopy(tehai1[value])
                #左から順に一枚ずつ仮に切っていく
                del tehai1[value]
                ags1.tehai_henkan(tehai1)
                #シャンテン数が増えたならもう一度最初からやる
                if ags1.shanten() > shantensuu:
                    continue
                #シャンテン数が変わらない場合(tehai1 は 13 枚)
                else:
                    #13 枚の時、仮に一枚入れてシャンテン数が進んだ牌を入れる
                    susumu_list = []
                    #シャンテン数が進む牌を探す
```

```
            for i in range(34):
                #仮りの手牌 2
                tehai2 = copy.deepcopy(tehai1)
                #仮りの手牌 2 に一つ牌を入れてみる
                tehai2.append(i)
                ags2.tehai_henkan(tehai2)
                #シャンテン数が進んだならリストに入れる
                if ags2.shanten() < ags1.shanten():
                    susumu_list.append(i)
            #手牌が進む残り牌の数を数えるための関数
            matihaisuu = int(len(susumu_list)) * 4
            #卓上で公になってる牌(自分の手牌,捨て牌,鳴いてる牌、ドラ)
            koukaihai = [jansi.tehai, taku.dora,
                         taku.jansi1_sutehai,
                         taku.jansi2_sutehai,
                         taku.jansi3_sutehai,
                         taku.jansi4_sutehai,
                         taku.jansi1_nakihai,
                         taku.jansi2_nakihai,
                         taku.jansi3_nakihai,
                         taku.jansi4_nakihai,]
            #手牌が進む残り牌の数を数える(減らしていく)
            for k in koukaihai:
                for i in k:
                    for j in susumu_list:
                        if i == j:
                            matihaisuu -= 1
            #haikouritu に[切った牌, 出た値]を入れる
            haikouritu.append([kiruhai ,matihaisuu])
    #テンパイなら積もった牌を捨てる
    if len(haikouritu) > 1:
        #待ち牌が多くなるものを選ぶ
        value2 = 0
        haikouritu2 = []
        for i in haikouritu:
            if i[1] > value2:
                haikouritu2 = [i[0]]
                value2 = i[1]
            elif i[1] == value2:
```

```
                haikouritu2.append(i[0])
        #東南西北→白發中→１・９牌→２・８牌→３・７牌→４・６牌→５牌と切っていく
        haikouritu3 = -1
        #if 文の階層を残して置くための関数
        count = 10
        if len(haikouritu2) > 0:
            for i in haikouritu2:
                if i == 27 or i == 28 or i == 29 or i == 30 and count > 0:
                    haikouritu3 = i
                    count = 1
                elif (i == 31 or i == 32 or i == 33) and count > 1:
                    haikouritu3 = i
                    count = 2
                elif (i == 0 or i == 8 or i == 9 or¥
                        i == 17 or i == 18 or i == 26) and count > 2:
                    haikouritu3 = i
                    count = 3
                elif (i == 1 or i == 7 or i == 10 or¥
                        i == 16 or i == 19 or i == 25) and count > 3:
                    haikouritu3 = i
                    count = 4
                elif (i == 2 or i == 6 or i == 11 or¥
                        i == 15 or i == 20 or i == 24) and count > 4:
                    haikouritu3 = i
                    count = 5
                elif (i == 3 or i == 5 or i == 12 or¥
                        i == 14 or i == 21 or i == 23) and count > 5:
                    haikouritu3 = i
                    count = 6
                elif count > 6:
                    haikouritu3 = i
                    count = 7
        #手牌から value に当てはまる牌の位置を覚える配列
        index = jansi.tehai.index(haikouritu3)
    else:
        #一番後ろを返す
        index = len(jansi.tehai) - 1
#降りると判断した時の打牌
else:
```

```
            index = jansi.tehai.index(orihai)
        #打牌する牌の場所を返す
        return index

#降りの実装
def ori(self, taku, jansi, shantensuu, teki, keikai):
        #全員のオリ牌
        orihai = []
        #合わせ打ち
        awase = []
        try:
            awase = list(set(jansi.tehai) & set(teki[2].get_sutehai()[-1]))
        except:
            pass
        if len(awase) > 0:
            orihai.append(teki[2].get_sutehai()[-1])
        #全員に共通の安全牌が 3 つあるものを抽出
        kanzen_anzenpai = list(set(teki[0].get_sutehai())&¥
                               set(teki[1].get_sutehai())&¥
                               set(teki[2].get_sutehai())    )
        #合わせ打ちできなかった場合
        if len(orihai) == 0:
            #警戒する敵がいない場合全員を降りる対象とする
            new_keikai = []
            if keikai == []:
                new_keikai = [teki[0],teki[1],teki[2]]
            else:
                new_keikai = copy.deepcopy(keikai)
            #警戒している敵の降り牌を入れる
            for i in new_keikai:
                #個人のオリ牌
                orihai2 = []
                #安全パイと同じ手牌があるならシャンテン数が少ない方を切る
                for j in jansi.tehai:
                    #安全パイがあるなら配列に入れる
                    try:
                        i.anzenpai.index(j)
                        orihai2.append(j)
                    except:
```

```
            pass
    #完全安全パイを残しておく
    for j in kanzen_anzenpai:
        while True:
            try:
                orihai2.remove(j)
            except:
                break
    #安全パイがなかったら完全安全パイを切る
    if len(orihai2) == 0:
        for j in kanzen_anzenpai:
            #安全パイがあるなら配列に入れる
            try:
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #完全安全パイがないなら単騎自牌をきる
    if len(orihai2) == 0:
        #現在見えている牌
        mieteru = jansi.tehai +¥
                  jansi.get_sutehai() + jansi.unit_nakihai() +¥
                  teki[0].get_sutehai() + teki[0].unit_nakihai() +¥
                  teki[1].get_sutehai() + teki[1].unit_nakihai() +¥
                  teki[2].get_sutehai() + teki[2].unit_nakihai()
        #3 枚見えてる牌を入れる
        sanmai = [x for x in set (mieteru) if mieteru.count(x) > 2]
        tankijihai = self.search_tankijihai(sanmai)
        for j in tankijihai:
            try:
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #単騎字牌がないならスジを切る
    if len(orihai2) == 0:
        suji = []
        #最初に表スジを降り牌に入れる
        for j in i.get_sutehai():
```

```
        if j == 3:
            suji += [0,6]
        if j == 4:
            suji += [1,7]
        if j == 5:
            suji += [2,8]
        if j == 12:
            suji += [9,15]
        if j == 13:
            suji += [10,16]
        if j == 14:
            suji += [11,17]
        if j == 21:
            suji += [18,24]
        if j == 22:
            suji += [19,25]
        if j == 23:
            suji += [20,26]
    #中スジをオリ牌に入れる
    nakasuji = list(set(i.get_sutehai()) & set([0,6]))
    if len(nakasuji) == 2:
        suji += [3]
    nakasuji = list(set(i.get_sutehai()) & set([1,7]))
    if len(nakasuji) == 2:
        suji += [4]
    nakasuji = list(set(i.get_sutehai()) & set([2,8]))
    if len(nakasuji) == 2:
        suji += [5]
    nakasuji = list(set(i.get_sutehai()) & set([9,15]))
    if len(nakasuji) == 2:
        suji += [12]
    nakasuji = list(set(i.get_sutehai()) & set([10,16]))
    if len(nakasuji) == 2:
        suji += [13]
    nakasuji = list(set(i.get_sutehai()) & set([11,17]))
    if len(nakasuji) == 2:
        suji += [14]
    nakasuji = list(set(i.get_sutehai()) & set([18,24]))
    if len(nakasuji) == 2:
```

```
            suji += [21]
        nakasuji = list(set(i.get_sutehai()) & set([19,25]))
        if len(nakasuji) == 2:
            suji += [22]
        nakasuji = list(set(i.get_sutehai()) & set([20,26]))
        if len(nakasuji) == 2:
            suji += [23]
        #スジ 19 を降り牌に入れる
        suji19 = [0,8,9,17,18,26]
        for j in suji19:
            try:
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #スジ 19 がないなら字牌をきる
    if len(orihai2) == 0:
        #字牌を降り牌に入れる
        jihai = [27,28,29,30,31,32,33]
        for j in jihai:
            try:
                suji.index(j)
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #スジ 28 を降り牌に入れる
    if len(orihai2) == 0:
        suji28 = [1,7,10,16,19,25]
        for j in suji28:
            try:
                suji.index(j)
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #スジ 37 を降り牌に入れる
    if len(orihai2) == 0:
        suji37 = [2,6,11,15,20,24]
```

```
        for j in suji37:
            try:
                suji.index(j)
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #無スジ 19 を降り牌に入れる
    if len(orihai2) == 0:
        musuji19 = [0,8,9,17,18,26]
        for j in musuji19:
            try:
                jansi.tehai.index(j)
                orihai2.append(j)
            except:
                pass
    #カタスジ 456
    if len(orihai2) == 0:
        #カタスジを切る
        for j in i.get_sutehai():
            if j == 0 or j == 6:
                suji += [3]
            if j == 1 or j == 7:
                suji += [4]
            if j == 2 or j == 8:
                suji += [5]
            if j == 9 or j == 15:
                suji += [12]
            if j == 10 or j == 16:
                suji += [13]
            if j == 11 or j == 17:
                suji += [14]
            if j == 18 or j == 24:
                suji += [21]
            if j == 19 or j == 25:
                suji += [22]
            if j == 20 or j == 26:
                suji += [23]
        #スジ 456 を降り牌に入れる
```

```
                suji456 = [3,4,5,12,13,14,21,22,23]
                for j in suji456:
                    try:
                        suji.index(j)
                        jansi.tehai.index(j)
                        orihai2.append(j)
                    except:
                        pass
            #無スジ 28 を降り牌に入れる
            if len(orihai2) == 0:
                musuji28 = [1,7,10,16,19,25]
                for j in musuji28:
                    try:
                        jansi.tehai.index(j)
                        orihai2.append(j)
                    except:
                        pass
            #無スジ 37 を降り牌に入れる
            if len(orihai2) == 0:
                musuji37 = [2,6,11,15,20,24]
                for j in musuji37:
                    try:
                        jansi.tehai.index(j)
                        orihai2.append(j)
                    except:
                        pass
            #無スジ 456 を降り牌に入れる
            if len(orihai2) == 0:
                musuji456 = [3,4,5,12,13,14,21,22,23]
                for j in musuji456:
                    try:
                        jansi.tehai.index(j)
                        orihai2.append(j)
                    except:
                        pass
            #全員のオリ牌に個人のオリ牌を入れる
            for j in orihai2:
                orihai.append(j)
    #降りる牌を確定する
```

```
        orihai3 = -1
        #２人以上リーチしているなら完全安全牌をだす
        if len(keikai) > 1 and len(kanzen_anzenpai) > 0:
            for j in kanzen_anzenpai:
                #安全パイがあるなら配列に入れる
                try:
                    jansi.tehai.index(j)
                    orihai3 = kanzen_anzenpai(j)
                    break
                except:
                    pass
        else:
            for j in collections.Counter(orihai).most_common():
                #一番降りれる牌を選ぶ
                try:
                    jansi.tehai.index(j)
                    orihai3 = j[0]
                    break
                except:
                    pass
        return orihai3

#リーチするかどうかの AI
def riichi(self, taku, jansi, teki):
        riichi = 'y'
        return riichi

#ロンするかどうかの AI
def ron(self):
        return 'y'

#暗カンするときの AI
def ankan(self):
        return 'n'

#明カンするときの AI
def minkan(self):
        return 'n'
```

```
#加カンするときの AI
def kakan(self):
    return 'y'

#ポンするときの AI
def pon(self, taku, jansi, sutehai, mae_ags, ato_ags):
    ponsuru = 'n'
    #捨て牌が役牌ならポンする
    if sutehai == 31 or sutehai == 32 or sutehai == 33 or¥
        (jansi.player_wind == EAST and sutehai == 27)or¥
        (jansi.player_wind == SOUTH and sutehai == 28)or¥
        (jansi.player_wind == WEST and sutehai == 29)or¥
        (jansi.player_wind == NORTH and sutehai == 30)or¥
        (taku.round_wind == EAST and sutehai == 27)or¥
        (taku.round_wind == SOUTH and sutehai == 28)or¥
        (taku.round_wind == WEST and sutehai == 29)or¥
        (taku.round_wind == NORTH and sutehai == 30):
        #シャンテン数が進んだらポンする(元から３枚あるならポンしない)
        if mae_ags > ato_ags:
            ponsuru = 'y'
    return ponsuru

#チーするときの AI
def chii(self, taku, jansi, sutehai, mae_ags, ato_ags):
    chiisuru = 'n'
    #すでに鳴いているなら鳴く
    if jansi.naki() == True:
        #シャンテン数が進んだらチーする
        if mae_ags > ato_ags:
            chiisuru = 'y'
    return chiisuru

#２パターン以上チーできる場合
def chii2(self, jansi, chii_value, ato_ags, ags):
    value = 0
    for i in chii_value:
        jansi.chii(i)
        ags.tehai_henkan(jansi.tehai)
        jansi.tehai.append(i[0])
```

```
                jansi.tehai.append(i[1])
                jansi.tehai.append(i[2])
                del jansi.chiihai[-1]
                if ato_ags == ags.shanten():
                    break
                value += 1
            return value

#要改善
    #降りるかどうかの判断
    def ori_judge(self, teki, taku, keikai, jansi, shantensuu, junni):
        judge = False
        #警戒している敵がいるなら降りる
        if len(keikai) > 0:
            judge = True

        #捨てた牌の合計が規定の数になったら降りる
        elif jansi.dahai_count <= 16:
            judge = True

        #自分が 4 位で一向聴なら降りない
        if junni[0] == 4 and shantensuu < 2:
            judge = False
        """
        #４位と点差がある場合のみオリない(自分 - 最下位)
        if junni[0] != 4:
            tensa = jansi.tenbou - junni[1].get_tenbou()
            if tensa <= 2000:
                judge = False
        ""
        #親ならオリない
        if jansi.player_wind == EAST:
            judge = False
        """
        return judge

    #警戒する敵を決める
    def keikai(self, teki, taku, jansi, junni):
        keikai = []
```

```
        #自分が親で４位の人がリーチした場合のみ、警戒する

        if junni[1].get_is_riichi() == True:
                keikai.append(junni[1])
        """
        #子の場合
        else:
        ""
        #他家がリーチしているなら
        for i in range(3):
            if teki[i].get_is_riichi() == True:
                keikai.append(teki[i])
        """
        return keikai

#単騎自牌を探す
def search_tankijihai(self, sanmai):
        tankijihai = []
        for i in sanmai:
            if i == 33:
                tankijihai.append(33)
            if i == 32:
                tankijihai.append(32)
            if i == 31:
                tankijihai.append(31)
            if i == 30:
                tankijihai.append(30)
            if i == 29:
                tankijihai.append(29)
            if i == 28:
                tankijihai.append(28)
            if i == 27:
                tankijihai.append(27)
        return tankijihai


#壁スジを探す
def search_kabesuji(self, yonmai):
```

```
kabesuji=[]
#萬子
if yonmai == 1:
    kabesuji = [0]
elif yonmai == 2:
    kabesuji = [0,1]
elif yonmai == 3:
    kabesuji = [1,2]
elif yonmai == 4:
    kabesuji = [2,6]
elif yonmai == 5:
    kabesuji = [6,7]
elif yonmai == 6:
    kabesuji = [7,8]
elif yonmai == 7:
    kabesuji = [8]
#筒子
elif yonmai == 10:
    kabesuji = [9]
elif yonmai == 11:
    kabesuji = [9,10]
elif yonmai == 12:
    kabesuji = [10,11]
elif yonmai == 13:
    kabesuji = [11,15]
elif yonmai == 14:
    kabesuji = [15,16]
elif yonmai == 15:
    kabesuji = [16,17]
elif yonmai == 16:
    kabesuji = [17]
#索子
elif yonmai == 19:
    kabesuji = [18]
elif yonmai == 20:
    kabesuji = [18,19]
elif yonmai == 21:
    kabesuji = [19,20]
elif yonmai == 22:
```

```
            kabesuji = [20,24]
        elif yonmai == 23:
            kabesuji = [24,25]
        elif yonmai == 24:
            kabesuji = [25,26]
        elif yonmai == 25:
            kabesuji = [26]

        return kabesuji

#自分の順位と最下位を調べる
def search_junni(self, taku, jansi, teki):
        #自分の順位
        jansi_junni = 4
        #[敵, 順位]を入れる
        saikai = None
        #自分の順位を入れる
        if taku.start_East == teki[0]:
            if jansi.tenbou > teki[0].get_tenbou():
                jansi_junni -= 1
            if jansi.tenbou > teki[1].get_tenbou():
                jansi_junni -= 1
            if jansi.tenbou > teki[2].get_tenbou():
                jansi_junni -= 1
        elif taku.start_East == teki[1]:
            if jansi.tenbou >= teki[0].get_tenbou():
                jansi_junni -= 1
            if jansi.tenbou > teki[1].get_tenbou():
                jansi_junni -= 1
            if jansi.tenbou > teki[2].get_tenbou():
                jansi_junni -= 1
        elif taku.start_East == teki[2]:
            if jansi.tenbou >= teki[0].get_tenbou():
                jansi_junni -= 1
            if jansi.tenbou >= teki[1].get_tenbou():
                jansi_junni -= 1
            if jansi.tenbou > teki[2].get_tenbou():
                jansi_junni -= 1
        else:
```

```
        if jansi.tenbou >= teki[0].get_tenbou():
            jansi_junni -= 1
        if jansi.tenbou >= teki[1].get_tenbou():
            jansi_junni -= 1
        if jansi.tenbou >= teki[2].get_tenbou():
            jansi_junni -= 1
    #最下位を決める
    #敵 1
    teki_junni1 = 4
    if taku.start_East == teki[0]:
        if teki[0].get_tenbou() >= teki[1].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() >= teki[2].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() >= jansi.tenbou:
            teki_junni1 -= 1
    elif taku.start_East == teki[1]:
        if teki[0].get_tenbou() > teki[1].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() > teki[2].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() > jansi.tenbou:
            teki_junni1 -= 1
    elif taku.start_East == teki[2]:
        if teki[0].get_tenbou() >= teki[1].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() > teki[2].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() > jansi.tenbou:
            teki_junni1 -= 1
    else:
        if teki[0].get_tenbou() >= teki[1].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() >= teki[2].get_tenbou():
            teki_junni1 -= 1
        if teki[0].get_tenbou() > jansi.tenbou:
            teki_junni1 -= 1
    if teki_junni1 == 4:
        saikai = teki[0]
```

```
#敵２
teki_junni2 = 4
if taku.start_East == teki[0]:
    if teki[1].get_tenbou() > teki[0].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() >= teki[2].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() >= jansi.tenbou:
        teki_junni2 -= 1
elif taku.start_East == teki[1]:
    if teki[1].get_tenbou() >= teki[0].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() >= teki[2].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() >= jansi.tenbou:
        teki_junni2 -= 1
elif taku.start_East == teki[2]:
    if teki[1].get_tenbou() > teki[0].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() > teki[2].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() > jansi.tenbou:
        teki_junni2 -= 1
else:
    if teki[1].get_tenbou() > teki[0].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() >= teki[2].get_tenbou():
        teki_junni2 -= 1
    if teki[1].get_tenbou() > jansi.tenbou:
        teki_junni2 -= 1
if teki_junni2 == 4:
    saikai = teki[1]
#敵３
teki_junni3 = 4
if taku.start_East == teki[0]:
    if teki[2].get_tenbou() > teki[0].get_tenbou():
        teki_junni3 -= 1
    if teki[2].get_tenbou() > teki[1].get_tenbou():
        teki_junni3 -= 1
```

```
            if teki[2].get_tenbou() >= jansi.tenbou:
                teki_junni3 -= 1
        if taku.start_East == teki[1]:
            if teki[2].get_tenbou() >= teki[0].get_tenbou():
                teki_junni3 -= 1
            if teki[2].get_tenbou() > teki[1].get_tenbou():
                teki_junni3 -= 1
            if teki[2].get_tenbou() >= jansi.tenbou:
                teki_junni3 -= 1
        if taku.start_East == teki[2]:
            if teki[2].get_tenbou() >= teki[0].get_tenbou():
                teki_junni3 -= 1
            if teki[2].get_tenbou() >= teki[1].get_tenbou():
                teki_junni3 -= 1
            if teki[2].get_tenbou() >= jansi.tenbou:
                teki_junni3 -= 1
        else:
            if teki[2].get_tenbou() > teki[0].get_tenbou():
                teki_junni3 -= 1
            if teki[2].get_tenbou() > teki[1].get_tenbou():
                teki_junni3 -= 1
            if teki[2].get_tenbou() > jansi.tenbou:
                teki_junni3 -= 1
        if teki_junni3 == 4:
            saikai = teki[2]
        return [jansi_junni, saikai]
```

<Computer.py>

```
# -*- coding: utf-8 -*-
from Taku import Taku
from Jansi import Jansi
from Agari_search import Agari_search
from mahjong.constants import EAST, SOUTH, WEST, NORTH
from Kanji import Kanji
from ComAI1 import ComAI1
import copy

#コンピューターの挙動
class Computer(object):
```

```
#コンピューターが麻雀をする時の挙動
def __init__(self, taku, name, comAI):
    self.jansi = Jansi()
    self.ags = Agari_search()
    self.taku = taku
    self.name = name
    #上がったとき用のフラグ
    self.agari_flag = False
    #テンパイ用のフラグ
    self.tenpai_flag = False
    #点棒のやり取り
    self.move_tenbou = []
    #自分の番が来る前に相手が鳴いていたかどうか
    self.naki = False
    #リーチ後に見逃しで振り聴になるフラグ
    self.huriten2 = False
    #AI
    self.comAI = comAI
    #自分以外の雀士の情報
    self.teki = []
    #自分の安全パイ
    self.anzenpai = []

#コンピュータが麻雀をするときの挙動
def game(self):
    print("=="*10,end='')
    #名前の表示
    self.show_name()
    print("=="*10)
    #自風牌の表示
    self.show_player_wind()
    print(":",end='')
    #点棒の表示
    self.show_tenbou()
    #リーチの表示
    if self.jansi.get_is_riichi() == True:
        print("")
        print("~~"*10,end='')
```

```
            print("立直",end='')
            print("~~"*10,end='')
        print("")
        print("")
        #捨て牌を表示
        self.jansi.show_sutehai()
        #理牌
        self.jansi.riipai()
        #ツモを得る
        self.jansi.tumo(self.taku.get_yama())
        #手牌をシャンテン数がわかるように変換する
        self.ags.tehai_henkan(self.jansi.get_tehai())
        #なき牌があるなら表示
        if len(self.jansi.get_ankanhai()) > 0 or len(self.jansi.get_minkanhai()) > 0 or¥
            len(self.jansi.get_ponhai()) > 0 or len(self.jansi.get_chiihai()) > 0:
            self.jansi.show_nakihai()
        #上がるかカンするときの処理
        self.agari_kan()
        #リーチをするかどうか、その後打牌する
        self.do_riichi()

#上がるかカンする時の処理
def agari_kan(self):
    while True:
        #上がれても役がないとダメ
        how_agari = False
        #上がるかどうかの処理
        if self.ags.shanten() < 0:
            #天和かどうか
            if self.jansi.dahai_count == 0 and self.jansi.ankanhai == [] and ¥
                                                    self.jansi.player_wind == EAST:
                self.jansi.is_tenhou = True
            #地和かどうか
            elif self.jansi.dahai_count == 0 and self.jansi.ankanhai == [] and ¥
                                                    self.jansi.player_wind != EAST:
                self.jansi.is_chiihou = True
            #嶺上開花かどうか
            if self.jansi.rinshanhai == self.jansi.tumohai:
                self.jansi.is_rinshan = True
```

```
#一発かどうか
if self.jansi.is_riichi == True and self.naki == False and¥
            self.jansi.riichi_dahai_count == 0:
        self.jansi.is_ippatsu = True
else:
        self.jansi.is_ippatsu = False
#海底かどうか
if len(self.taku.get_yama()) == 14:
        self.jansi.is_haitei = True
#上がるための流れ(全部必要)
self.jansi.unit_agari()
self.ags.tehai_henkan(self.jansi.get_unit_tehai())
#ツモ判定
if self.jansi.naki() == False:
        self.jansi.set_is_tsumo(True)
else:
        self.jansi.set_is_tsumo(False)
try:
        self.move_tenbou = self.calculation()
        how_agari = True
except:
        pass
if how_agari == True:
        #上がったときの表示
        print(self.name + "はツモりました")
        self.ags.print_hand_result()
        self.agari_flag = True
        self.add_tenbou2()
        #手牌の表示
        self.show_tehai()
        #なき牌の表示
        self.jansi.show_nakihai()
        #プレイヤーを表示
        self.show_name()
        #持ち点を表示
        self.jansi.show_tenbou()
        print("")
        break
else:
```

```
                    #ツモ判定をなくす
                    self.jansi.set_is_tsumo(False)
                    #リンシャン判定をなくす
                    self.jansi.set_is_rinshan(False)
                    self.jansi.rinshanhai = -1
                print("")
                #牌が 4 つあるものを抽出
                kan_value = [x for x in set (self.jansi.get_tehai())
                                    if self.jansi.get_tehai().count(x) > 3]
                #カンするかどうか
                if len(kan_value) > 0 and len(self.taku.get_yama()) > 14:
#未実装(int)
                    self.comAI.ankan()

                    kansuru = 'n'
                    if kansuru == 'y':
                        self.jansi.ankan(kan_value, self.taku)
                        #暗カンした牌の表示
                        print(self.name + "は" +¥
                                    Kanji().get_kanji(self.jansi.ankanhai[-1][0])+ "を暗カンしました。")
                        #ドラを表示
                        self.taku.show_dora()
                        #もう一度カンや上がれる場合ループする
                        if    len(kan_value) > 0 or self.ags.shanten() < 0:
                            continue
                #カンしない(できない)時や和了しない時はループを抜ける
                break

    #リーチするかどうか、その後打牌する
    def do_riichi(self):
        if self.agari_flag == False:
            #リーチしてるならツモ切り
            if self.jansi.get_is_riichi() == True:
                self.jansi.dahai(len(self.jansi.get_tehai())-1)
                #リーチの時の打牌した数をカウント
                self.jansi.riichi_dahai_count += 1
                #打牌した牌の表示
                print(self.name + "は" +¥
                            Kanji().get_kanji(self.jansi.get_sutehai()[-1])+ "を切りました。")
```

```
#あとで消す
                    self.show_tehai()
                else:
                    if self.ags.shanten() == 0 and self.jansi.naki() == False:
                        self.comAI.riichi(self.taku, self.jansi, self.teki)
                        riichi1 = 'y'
                        if riichi1 == 'y':
                            #リー棒を場に置く
                            self.jansi.sub_tenbou(1000)
                            self.taku.set_oki_riibou()
                            self.jansi.set_is_riichi()
                            #ダブルリーチかどうか
                            if self.jansi.dahai_count == 0:
                                self.jansi.is_daburu_riichi = True
                            #リーチした牌の表示
                            print(self.name + "はリーチしました。")
                    #どの牌を切るか
                    self.dahai()

    #どの牌を切るかの処理
    def dahai(self):
        self.ags.tehai_henkan(self.jansi.tehai)
        value = self.comAI.dahai(self.taku, self.jansi, self.ags.shanten(), self.teki)



        self.jansi.dahai(int(value))
        #リンシャン等の判定をリセット
        self.jansi.reset_yaku()
        #自分の番の後、他が鳴いたかどうかのフラグをリセット
        self.naki = False
        #打牌した数をカウント
        self.jansi.dahai_count += 1
        #打牌した牌の表示
        print(self.name + "は" +¥
                    Kanji().get_kanji(self.jansi.get_sutehai()[-1])+ "を切りました。")

    # 1 局を終わるときの処理
    def finish_round(self, dareka_agari):
        nagasi = False
```

```
#誰も上がっていないなら
if dareka_agari == False:
    self.ags.tehai_henkan(self.jansi.get_tehai())
    #テンパイかどうか
    if self.ags.shanten() <= 0:
        print(self.name + "はテンパイでした")
        #手牌の表示
        self.show_tehai()
        self.tenpai_flag = True
    else:
        print(self.name + "はノーテンでした")
        self.tenpai_flag = False
    #流し満貫かどうか
    nagasi = True
    for i in self.jansi.sutehai:
        if i == 0 or i ==8 or i ==9 or i ==17 or i ==18 or i ==26 or i ==27¥
                or i ==28 or i ==29 or i ==30 or i ==31 or i ==32 or i ==33:
            pass
        else:
            nagasi = False
    #流し満貫なら,ノーテンでも和了する
    if nagasi == True:
        print(self.name + "は流し満貫でした")
        self.jansi.is_nagashi_mangan = True
        #上がるための流れ(全部必要)
        self.jansi.unit_agari()
        self.ags.tehai_henkan(self.jansi.get_unit_tehai())
        self.move_tenbou = self.calculation()
        self.add_tenbou2()
        self.ags.print_hand_result()
#牌をリセットする
self.jansi.reset_hai()
#役をリセットする
self.jansi.reset_yaku()
#リーチの役をリセットする
self.jansi.is_riichi = False
#自分の番の後、他が鳴いたかどうかのフラグをリセット
self.naki = False
#リーチ後に見逃しで振り聴になるフラグ
```

```
        self.huriten2 = False
        return nagasi

#ロンの処理
def ron(self, sutehai, teki_name):
        #捨て牌を手牌に加える
        self.jansi.get_tehai().append(sutehai)
        self.ags.tehai_henkan(self.jansi.get_tehai())
        #上がれても役がないとダメ
        how_agari = False
        #上がるかどうかの処理(役があるかどうか)
        if self.ags.shanten() < 0:
                #捨てた牌をツモ牌に設定する
                self.jansi.set_tumohai(sutehai)
                #人和かどうか
                if self.jansi.dahai_count == 0 and self.naki == False¥
                        and self.jansi.ankanhai == []:
                        self.jansi.is_renhou = True
                #一発かどうか
                if self.jansi.is_riichi == True and self.naki == False and¥
                                self.jansi.riichi_dahai_count == 1:
                        self.jansi.is_ippatsu = True
                else:
                        self.jansi.is_ippatsu = False
                #河底かどうか
                if len(self.taku.get_yama()) == 14:
                        self.jansi.is_houtei = True
                #上がるための流れ(全部必要)
                self.jansi.unit_agari()
                self.ags.tehai_henkan(self.jansi.get_unit_tehai())
                try:
                        self.move_tenbou = self.calculation()
                        how_agari = True
                except:
                        pass
        if how_agari == False:
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]
        elif how_agari == True:
```

```
                #振り聴かどうか
                huriten = False
                #上がり牌の検索
                agarihai = self.search_agarihai()
                #もしロンしようとした手が振り聴だったら上がれない
                for j in agarihai:
                    for i in self.jansi.get_sutehai():
                        if i == j:
                            huriten = True
                #振り聴じゃない場合
                if huriten == False and self.huriten2 == False:
#未実装
                    self.comAI.ron()
                    agaru = 'y'

                    if agaru == 'y':
                        #上がったときの表示
                        print(self.name + "は" + teki_name +"から上がりました")
                        #上がりの表示
                        self.ags.print_hand_result()
                        self.agari_flag = True
                        self.add_tenbou2()
                        #手牌の表示
                        self.show_tehai()
                        #プレイヤーを表示
                        self.show_name()
                        #なき牌の表示
                        self.jansi.show_nakihai()
                        #持ち点を表示
                        self.jansi.show_tenbou()
                        print("")
                        self.agari_flag = True
                    elif agaru == 'n':
                        #一度手に加えた牌を捨てる
                        del self.jansi.get_tehai()[-1]
                        #一度見逃したら振り聴
                        if self.jansi.is_riichi == True:
                            self.huriten2 = True
                elif huriten == True:
```

```
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]

    #ミンカンの処理
    def minkan(self, sutehai):
        if self.jansi.get_is_riichi() == False:
            #捨て牌を手牌に加える
            self.jansi.get_tehai().append(sutehai)
            #牌が 4 つあるものを抽出
            kan_value = [x for x in set (self.jansi.get_tehai())
                                    if self.jansi.get_tehai().count(x) > 3]
            #カンするかどうか
            if len(kan_value) > 0 and len(self.taku.get_yama()) > 14:
                kansuru = 'n'
                for i in kan_value:
                    if i == sutehai:
#未実装
                        self.comAI.minkan()
                        kansuru = 'n'

                        if kansuru == 'y':
                            #明カンする
                            self.jansi.minkan(sutehai, self.taku)
                            #明カンした牌の表示
                            print(self.name + "は" +¥
                                        Kanji().get_kanji(self.jansi.minkanhai[-1][0])+ "を明カンしました。
")
                            #リンシャン牌をツモ牌に設定する(リンシャンのため)
                            self.jansi.rinshanhai = copy.deepcopy(self.jansi.tumohai)
                            #上がるかカンするか
                            self.agari_kan()
                            #打牌
                            self.dahai()
                            #ドラを表示
                            self.taku.show_dora()
                            #カンの回数を増やす
                            self.taku.add_kan()
                            #表のドラの数を増やす(明カンなのでリンシャンを得る後に)
                            self.taku.add_dora()
```

```
                    return True
            if kansuru == 'n':
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]
        else:
            #一度手に加えた牌を捨てる
            del self.jansi.get_tehai()[-1]
    return False

#ポンの処理
def pon(self, sutehai):
    if self.jansi.get_is_riichi() == False:
        #捨て牌を手牌に加える
        self.jansi.get_tehai().append(sutehai)
        #牌が 3 つあるものを抽出
        pon_value = [x for x in set (self.jansi.get_tehai())
                            if self.jansi.get_tehai().count(x) > 2]
        #ポンするかどうか
        if len(pon_value) > 0:
            ponsuru='n'
            for i in pon_value:
                #手牌の 3 つある牌とさっき捨てた牌が一致しているならポンできる
                if i == sutehai:
                    #一度手に加えた牌を捨てる(手牌を表示する時にわかりやすくするため)
                    del self.jansi.get_tehai()[-1]
                    self.show_tehai()
                    #手牌に加える前のシャンテン数
                    self.ags.tehai_henkan(self.jansi.tehai)
                    mae_ags = self.ags.shanten()
                    #捨て牌をもう一度手牌に加える
                    self.jansi.get_tehai().append(sutehai)
                    #手牌に加えた後のシャンテン数
                    self.ags.tehai_henkan(self.jansi.tehai)
                    ato_ags = self.ags.shanten()
                    #ポンするかどうか
                    ponsuru = self.comAI.pon(self.taku,self.jansi, sutehai, mae_ags, ato_ags)
                    if ponsuru == 'y':
                        self.jansi.pon(sutehai)
                        #ポンした牌の表示
```

```
                print(self.name + "は" +¥
                        Kanji().get_kanji(self.jansi.ponhai[-1][0])+ "をポンしました。")
                self.dahai()
                return True
        if ponsuru == 'n':
            #一度手に加えた牌を捨てる
            del self.jansi.get_tehai()[-1]
    else:
        #一度手に加えた牌を捨てる
        del self.jansi.get_tehai()[-1]
return False

#チーの処理は一番下にある（長いので）

#カカン
def kakan(self):
    for i in range(len(jansi.ponhai)):
        #ポン牌と同じものを抽出
        kan_value = tehai.index(jansi.ponhai[i][0])
    #カンするかどうか
    if len(kan_value) > 0 and len(taku.get_yama()) > 14:
        kansuru = 'n'
        for i in kan_value:
            kansuru = self.comAI().kakan()
            if kansuru == 'y':
                #加カンする
                self.jansi.kakan(kan_value, self.taku)
                #加カンした牌の表示
                print(self.name + "は" +¥
                        Kanji().get_kanji(self.jansi.minkanhai[-1][0])+ "を加カンしました。")
                #リンシャン牌をツモ牌に設定する(リンシャンのため)
                self.jansi.rinshanhai = copy.deepcopy(self.jansi.tumohai)
                #手牌を見せる
                self.show_tehai()
                #上がるかカンするか
                self.agari_kan()
                #打牌
                self.dahai()
                #ドラを表示
```

```
                    self.taku.show_dora()
                    #カンの回数を増やす
                    self.taku.add_kan()
                    #表のドラの数を増やす(明カンなのでリンシャンを得る後に)
                    self.taku.add_dora()

                    return True
            if kansuru == 'n':
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]
        else:
            #一度手に加えた牌を捨てる
            del self.jansi.get_tehai()[-1]
        return False

#捨て牌のゲッター
def get_sutehai(self):
    return self.jansi.get_sutehai()

#立直のゲッター
def get_is_riichi(self):
    return self.jansi.is_riichi

#自風のゲッター
def get_player_wind(self):
    return self.jansi.get_player_wind()

#やり取りする点棒のゲッター
def get_move_tenbou(self):
    return self.move_tenbou

#点棒を加える(テンパイ時)
def add_tenbou(self, tenbou):
    self.jansi.add_tenbou(tenbou)

#点棒を加える
def add_tenbou2(self):
    tenbou2 = self.move_tenbou[0] + self.move_tenbou[1]*2 + self.taku.get_honba()*300
    #場のリーチ棒も持っていく
```

```
        self.jansi.add_tenbou(tenbou2 + self.taku.get_oki_riibou())
        self.taku.oki_riibou = 0


#点棒を減らす
def sub_tenbou(self, tenbou2):
        self.jansi.sub_tenbou(tenbou2)

#点棒
def get_tenbou(self):
        return self.jansi.tenbou

#テンパイしてたかどうかのフラグのゲッター
def get_tenpai_flag(self):
        return self.tenpai_flag

#上がるためのフラグのゲッター
def get_agari_flag(self):
        return self.agari_flag

#上がりフラグのリセット
def reset_agari_flag(self):
        self.agari_flag = False

#点棒の表示
def show_tenbou(self):
        self.jansi.show_tenbou()

#点棒の表示
def show_tenbou(self):
        self.jansi.show_tenbou()

#自風牌の表示
def show_player_wind(self):
        self.jansi.show_player_wind()

#自風牌を変える
def change_player_wind(self):
        self.jansi.change_player_wind()
```

```
#名前の表示
def show_name(self):
    print(" [" + self.name + "] ",end='')

#自風のセッター
def set_player_wind(self, player_wind):
    self.jansi.set_player_wind(player_wind)

#手牌に牌を加える
def append_hai(self, hai):
    self.jansi.get_tehai().append(hai)

#配牌
def haipai(self):
    self.jansi.haipai(self.taku.get_yama())

#理牌
def riipai(self):
    self.jansi.riipai()

#点棒の計算
def calculation(self):
    return self.ags.agari(self.jansi.get_tumohai(),
                          self.taku.get_dora(),
                          self.jansi.get_is_tsumo(),
                          self.jansi.get_is_riichi(),
                          self.jansi.get_is_ippatsu(),
                          self.jansi.get_is_rinshan(),
                          self.jansi.get_is_chankan(),
                          self.jansi.get_is_haitei(),
                          self.jansi.get_is_houtei(),
                          self.jansi.get_is_daburu_riichi(),
                          self.jansi.get_is_nagashi_mangan(),
                          self.jansi.get_is_tenhou(),
                          self.jansi.get_is_renhou(),
                          self.jansi.get_is_chiihou(),
                          self.jansi.get_player_wind(),
                          self.taku.get_round_wind(),
```

```
                    self.jansi.get_ankanhai(),
                    self.jansi.get_minkanhai(),
                    self.jansi.get_ponhai(),
                    self.jansi.get_chiihai())

#手牌を漢字に変換して見せる
def show_tehai(self):
    bangou = ["⓪","①","②","③","④","⑤","⑥",
              "⑦","⑧","⑨","⑩","⑪","⑫","⑬"]
    for i in range(len(self.jansi.get_tehai())):
        print(" " + str(bangou[i]) + "    ",end='')
    print("")
    for i in self.jansi.get_tehai():
        print(Kanji().get_kanji(i),end='')
    print("")

#ツモ牌を漢字に変換して見せる
def show_tumo(self):
    try:
        if self.tehai[13] is not None:
            print("ツモ: " + self.kanji.get_kanji(self.tehai[13]))
    except:
        pass

#チーの処理
def chii(self, sutehai):
    if self.jansi.get_is_riichi() == False:
        #順子を入れる
        chii_value = []
        #萬子
        if sutehai == 0:
            try:
                if self.jansi.get_tehai().index(1) is not None¥
                    and self.jansi.get_tehai().index(2) is not None:
                    chii_value.append([0,1,2])
            except:
                pass
        elif sutehai == 1:
            try:
```

```
            if self.jansi.get_tehai().index(0) is not None¥
                and self.jansi.get_tehai().index(2) is not None:
                chii_value.append([0,1,2])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(2) is not None¥
                and self.jansi.get_tehai().index(3) is not None:
                chii_value.append([1,2,3])
        except:
            pass
    elif sutehai == 2:
        try:
            if self.jansi.get_tehai().index(0) is not None¥
                and self.jansi.get_tehai().index(1) is not None:
                chii_value.append([0,1,2])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(1) is not None¥
                and self.jansi.get_tehai().index(3) is not None:
                chii_value.append([1,2,3])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(3) is not None¥
                and self.jansi.get_tehai().index(4) is not None:
                chii_value.append([2,3,4])
        except:
            pass
    elif sutehai == 3:
        try:
            if self.jansi.get_tehai().index(1) is not None¥
                and self.jansi.get_tehai().index(2) is not None:
                chii_value.append([1,2,3])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(2) is not None¥
```

```
                and self.jansi.get_tehai().index(4) is not None:
                chii_value.append([2,3,4])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(4) is not None¥
                and self.jansi.get_tehai().index(5) is not None:
                chii_value.append([3,4,5])
        except:
            pass
    elif sutehai == 4:
        try:
            if self.jansi.get_tehai().index(2) is not None¥
                and self.jansi.get_tehai().index(3) is not None:
                chii_value.append([2,3,4])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(3) is not None¥
                and self.jansi.get_tehai().index(5) is not None:
                chii_value.append([3,4,5])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(5) is not None¥
                and self.jansi.get_tehai().index(6) is not None:
                chii_value.append([4,5,6])
        except:
            pass
    elif sutehai == 5:
        try:
            if self.jansi.get_tehai().index(3) is not None¥
                and self.jansi.get_tehai().index(4) is not None:
                chii_value.append([3,4,5])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(4) is not None¥
                and self.jansi.get_tehai().index(6) is not None:
```

```
                chii_value.append([4,5,6])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(6) is not None¥
                and self.jansi.get_tehai().index(7) is not None:
                chii_value.append([5,6,7])
        except:
            pass
    elif sutehai == 6:
        try:
            if self.jansi.get_tehai().index(4) is not None¥
                and self.jansi.get_tehai().index(5) is not None:
                chii_value.append([4,5,6])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(5) is not None¥
                and self.jansi.get_tehai().index(7) is not None:
                chii_value.append([5,6,7])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(7) is not None¥
                and self.jansi.get_tehai().index(8) is not None:
                chii_value.append([6,7,8])
        except:
            pass
    elif sutehai == 7:
        try:
            if self.jansi.get_tehai().index(5) is not None¥
                and self.jansi.get_tehai().index(6) is not None:
                chii_value.append([5,6,7])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(6) is not None¥
                and self.jansi.get_tehai().index(8) is not None:
                chii_value.append([6,7,8])
```

```
        except:
            pass
    elif sutehai == 8:
        try:
            if self.jansi.get_tehai().index(6) is not None¥
                and self.jansi.get_tehai().index(7) is not None:
                chii_value.append([6,7,8])
        except:
            pass
    #筒子
    elif sutehai == 9:
        try:
            if self.jansi.get_tehai().index(10) is not None¥
                and self.jansi.get_tehai().index(11) is not None:
                chii_value.append([9,10,11])
        except:
            pass
    elif sutehai == 10:
        try:
            if self.jansi.get_tehai().index(9) is not None¥
                and self.jansi.get_tehai().index(11) is not None:
                chii_value.append([9,10,11])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(11) is not None¥
                and self.jansi.get_tehai().index(12) is not None:
                chii_value.append([10,11,12])
        except:
            pass
    elif sutehai == 11:
        try:
            if self.jansi.get_tehai().index(9) is not None¥
                and self.jansi.get_tehai().index(10) is not None:
                chii_value.append([9,10,11])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(10) is not None¥
```

```
                and self.jansi.get_tehai().index(12) is not None:
                chii_value.append([10,11,12])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(12) is not None¥
                and self.jansi.get_tehai().index(13) is not None:
                chii_value.append([11,12,13])
        except:
            pass
    elif sutehai == 12:
        try:
            if self.jansi.get_tehai().index(10) is not None¥
                and self.jansi.get_tehai().index(11) is not None:
                chii_value.append([10,11,12])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(11) is not None¥
                and self.jansi.get_tehai().index(13) is not None:
                chii_value.append([11,12,13])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(13) is not None¥
                and self.jansi.get_tehai().index(14) is not None:
                chii_value.append([12,13,14])
        except:
            pass
    elif sutehai == 13:
        try:
            if self.jansi.get_tehai().index(11) is not None¥
                and self.jansi.get_tehai().index(12) is not None:
                chii_value.append([11,12,13])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(12) is not None¥
                and self.jansi.get_tehai().index(14) is not None:
```

```
                chii_value.append([12,13,14])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(14) is not None¥
                and self.jansi.get_tehai().index(15) is not None:
                chii_value.append([13,14,15])
        except:
            pass
    elif sutehai == 14:
        try:
            if self.jansi.get_tehai().index(12) is not None¥
                and self.jansi.get_tehai().index(13) is not None:
                chii_value.append([12,13,14])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(13) is not None¥
                and self.jansi.get_tehai().index(15) is not None:
                chii_value.append([13,14,15])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(15) is not None¥
                and self.jansi.get_tehai().index(16) is not None:
                chii_value.append([14,15,16])
        except:
            pass
    elif sutehai == 15:
        try:
            if self.jansi.get_tehai().index(13) is not None¥
                and self.jansi.get_tehai().index(14) is not None:
                chii_value.append([13,14,15])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(14) is not None¥
                and self.jansi.get_tehai().index(16) is not None:
                chii_value.append([14,15,16])
```

```
        except:
            pass
        try:
            if self.jansi.get_tehai().index(16) is not None¥
                and self.jansi.get_tehai().index(17) is not None:
                chii_value.append([15,16,17])
        except:
            pass
    elif sutehai == 16:
        try:
            if self.jansi.get_tehai().index(14) is not None¥
                and self.jansi.get_tehai().index(15) is not None:
                chii_value.append([14,15,16])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(15) is not None¥
                and self.jansi.get_tehai().index(17) is not None:
                chii_value.append([15,16,17])
        except:
            pass
    elif sutehai == 17:
        try:
            if self.jansi.get_tehai().index(15) is not None¥
                and self.jansi.get_tehai().index(16) is not None:
                chii_value.append([15,16,17])
        except:
            pass
    #索子
    elif sutehai == 18:
        try:
            if self.jansi.get_tehai().index(19) is not None¥
                and self.jansi.get_tehai().index(20) is not None:
                chii_value.append([18,19,20])
        except:
            pass
    elif sutehai == 19:
        try:
            if self.jansi.get_tehai().index(18) is not None¥
```

```
                and self.jansi.get_tehai().index(20) is not None:
                chii_value.append([18,19,20])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(20) is not None¥
                and self.jansi.get_tehai().index(21) is not None:
                chii_value.append([19,20,21])
        except:
            pass
    elif sutehai == 20:
        try:
            if self.jansi.get_tehai().index(18) is not None¥
                and self.jansi.get_tehai().index(19) is not None:
                chii_value.append([18,19,20])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(19) is not None¥
                and self.jansi.get_tehai().index(21) is not None:
                chii_value.append([19,20,21])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(21) is not None¥
                and self.jansi.get_tehai().index(22) is not None:
                chii_value.append([20,21,22])
        except:
            pass
    elif sutehai == 21:
        try:
            if self.jansi.get_tehai().index(19) is not None¥
                and self.jansi.get_tehai().index(20) is not None:
                chii_value.append([19,20,21])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(20) is not None¥
                and self.jansi.get_tehai().index(22) is not None:
```

```
                chii_value.append([20,21,22])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(22) is not None¥
                and self.jansi.get_tehai().index(23) is not None:
                chii_value.append([21,22,23])
        except:
            pass
    elif sutehai == 22:
        try:
            if self.jansi.get_tehai().index(20) is not None¥
                and self.jansi.get_tehai().index(21) is not None:
                chii_value.append([20,21,22])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(21) is not None¥
                and self.jansi.get_tehai().index(23) is not None:
                chii_value.append([21,22,23])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(23) is not None¥
                and self.jansi.get_tehai().index(24) is not None:
                chii_value.append([22,23,24])
        except:
            pass
    elif sutehai == 23:
        try:
            if self.jansi.get_tehai().index(21) is not None¥
                and self.jansi.get_tehai().index(22) is not None:
                chii_value.append([21,22,23])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(22) is not None¥
                and self.jansi.get_tehai().index(24) is not None:
                chii_value.append([22,23,24])
```

```
        except:
            pass
        try:
            if self.jansi.get_tehai().index(24) is not None¥
                and self.jansi.get_tehai().index(25) is not None:
                chii_value.append([23,24,25])
        except:
            pass
    elif sutehai == 24:
        try:
            if self.jansi.get_tehai().index(22) is not None¥
                and self.jansi.get_tehai().index(23) is not None:
                chii_value.append([22,23,24])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(23) is not None¥
                and self.jansi.get_tehai().index(25) is not None:
                chii_value.append([23,24,25])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(25) is not None¥
                and self.jansi.get_tehai().index(26) is not None:
                chii_value.append([24,25,26])
        except:
            pass
    elif sutehai == 25:
        try:
            if self.jansi.get_tehai().index(23) is not None¥
                and self.jansi.get_tehai().index(24) is not None:
                chii_value.append([23,24,25])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(24) is not None¥
                and self.jansi.get_tehai().index(26) is not None:
                chii_value.append([24,25,26])
        except:
```

```
            pass
    elif sutehai == 26:
        try:
            if self.jansi.get_tehai().index(24) is not None¥
                and self.jansi.get_tehai().index(25) is not None:
                chii_value.append([24,25,26])
        except:
            pass
    #チーするかどうか
    if len(chii_value) > 0:
        chiisuru = 'n'
        self.show_tehai()
        #手牌に加える前のシャンテン数
        self.ags.tehai_henkan(self.jansi.tehai)
        mae_ags = self.ags.shanten()
        #捨て牌をもう一度手牌に加える
        self.jansi.get_tehai().append(sutehai)
        #手牌に加えた後のシャンテン数
        self.ags.tehai_henkan(self.jansi.tehai)
        ato_ags = self.ags.shanten()
        chiisuru = self.comAI.chii(self.taku,self.jansi, sutehai, mae_ags, ato_ags)
        if chiisuru == 'y':
            #チーできる牌の最初の組み合わせを選ぶ
            dore = 0
            #２個以上チーができるなら
            if len(chii_value) > 1:
                dore = self.comAI.chii2(self.jansi, chii_value, ato_ags, self.ags)
            self.show_tehai()
            #理牌する
            self.riipai()
            self.jansi.chii(chii_value[int(dore)])
            #チーした牌の表示
            print(self.name + "は" +¥
                    Kanji().get_kanji(self.jansi.chiihai[-1][0])+¥
                    Kanji().get_kanji(self.jansi.chiihai[-1][1])+¥
                    Kanji().get_kanji(self.jansi.chiihai[-1][2])+¥
                     "をチーしました。")
            self.dahai()
            #捨て牌を回収する
```

```
                    del sutehai
                    return True
                else:
                    del self.jansi.get_tehai()[-1]
        return False

#なき牌を返す
def unit_nakihai(self):
        return self.jansi.unit_nakihai()

#上がり牌の検索
def search_agarihai(self):
        #仮りの上がりの検索
        ags2 = Agari_search()
        agari_list = []
        #上がり牌を探す
        for i in range(34):
            #仮りの手牌
            tehai2 = copy.deepcopy(self.jansi.get_tehai())
            del tehai2[-1]
            #仮りの手牌に一つ牌を入れてみる
            tehai2.append(i)
            ags2.tehai_henkan(tehai2)
            #上がり牌ならリストに入れる
            if ags2.shanten() < 0:
                agari_list.append(i)
        print(agari_list)
        #上がり牌を返す
        return agari_list

#敵をセットする
def set_teki(self,teki):
        self.teki = teki

#安全パイをセットする
def set_anzenpai(self,anzenpai):
        self.anzenpai = anzenpai
```

<Human.py>

```
# -*- coding: utf-8 -*-
from Taku import Taku
from Jansi import Jansi
from Agari_search import Agari_search
from mahjong.constants import EAST, SOUTH, WEST, NORTH
from Kanji import Kanji
import copy

#人間の挙動
class Human(object):

    #人間が麻雀をする時の挙動
    def __init__(self, taku, name):
        self.jansi = Jansi()
        self.ags = Agari_search()
        self.taku = taku
        self.name = name
        #上がったとき用のフラグ
        self.agari_flag = False
        #テンパイ用のフラグ
        self.tenpai_flag = False
        #点棒のやり取り
        self.move_tenbou = []
        #自分の番が来る前に相手が鳴いていたかどうか
        self.naki = False
        #リーチ後に見逃しで振り聴になるフラグ
        self.huriten2 = False

    #人間が麻雀をするときの挙動
    def game(self):
        print("=="*10,end='')
        #名前の表示
        self.show_name()
        print("=="*10)
        #自風牌の表示
        self.show_player_wind()
        print(":",end='')
        #点棒の表示
```

```
        self.show_tenbou()
        #リーチの表示
        if self.jansi.get_is_riichi() == True:
            print("")
            print("~~"*10,end='')
            print("立直",end='')
            print("~~"*10,end='')
        print("")
        print("")
        #捨て牌を表示
        self.jansi.show_sutehai()
        #理牌
        self.jansi.riipai()
        #ツモを得る
        self.jansi.tumo(self.taku.get_yama())
        #手牌をシャンテン数がわかるように変換する
        self.ags.tehai_henkan(self.jansi.get_tehai())
        #ツモと手牌の表示
        self.show_tumo()
        self.show_tehai()
        print("シャンテン数: " + str(int(self.ags.shanten())))
        #なき牌があるなら表示
        if len(self.jansi.get_ankanhai()) > 0 or len(self.jansi.get_minkanhai()) > 0 or¥
            len(self.jansi.get_ponhai()) > 0 or len(self.jansi.get_chiihai()) > 0:
             self.jansi.show_nakihai()
        #上がるかカンするときの処理
        self.agari_kan()
        #リーチをするかどうか、その後打牌する
        self.do_riichi()

#上がるかカンする時の処理
def agari_kan(self):
        while True:
            #上がれても役がないとダメ
            how_agari = False
            #上がるかどうかの処理
            if self.ags.shanten() < 0:
                #天和かどうか
                if self.jansi.dahai_count == 0 and self.jansi.ankanhai == [] and ¥
```

```
                                                self.jansi.player_wind == EAST:
        self.jansi.is_tenhou = True
#地和かどうか
elif self.jansi.dahai_count == 0 and self.jansi.ankanhai == [] and ¥
                                                self.jansi.player_wind != EAST:
        self.jansi.is_chiihou = True
#嶺上開花かどうか
if self.jansi.rinshanhai == self.jansi.tumohai:
        self.jansi.is_rinshan = True
#一発かどうか
if self.jansi.is_riichi == True and self.naki == False and¥
            self.jansi.riichi_dahai_count == 0:
        self.jansi.is_ippatsu = True
else:
        self.jansi.is_ippatsu = False
#海底かどうか
if len(self.taku.get_yama()) == 14:
        self.jansi.is_haitei = True
#上がるための流れ(全部必要)
self.jansi.unit_agari()
self.ags.tehai_henkan(self.jansi.get_unit_tehai())
#ツモ判定
if self.jansi.naki() == False:
        self.jansi.set_is_tsumo(True)
else:
        self.jansi.set_is_tsumo(False)
try:
        self.move_tenbou = self.calculation()
        how_agari = True
except:
        print("役がありません")
        pass
if how_agari == True:
        while True:
            try:
                agaru = input("上がりますか？  y/n: ")
                if agaru == 'y' or agaru == 'n':
                    break
            except:
```

```
                    pass
            if agaru == 'y':
                self.ags.print_hand_result()
                self.agari_flag = True
                self.add_tenbou2()
                #手牌の表示
                self.show_tehai()
                #持ち点を表示
                self.jansi.show_tenbou()
                break
            else:
                #ツモ判定をなくす
                self.jansi.set_is_tsumo(False)
                #リンシャン判定をなくす
                self.jansi.set_is_rinshan(False)
                self.jansi.rinshanhai = -1
print("")
#牌が 4 つあるものを抽出
kan_value = [x for x in set (self.jansi.get_tehai())
                        if self.jansi.get_tehai().count(x) > 3]
#カンするかどうか
if len(kan_value) > 0 and len(self.taku.get_yama()) > 14:
    while True:
        try:
            kansuru = input("暗カンしますか？(y/n): ")
            if kansuru == 'y' or kansuru == 'n':
                break
        except:
            pass
    if kansuru == 'y':
        #もし牌が 4 つあるものが２種類以上あるなら
        while True:
            try:
                print("どの牌を暗カンしますか?(左から 0,1...)")
                kan00 = []
                for i in range(len(kan_value)):
                    kan00.append(Kanji().get_kanji(kan_value[i]))
                print(kan00,end = '')
                kan_index = input("：　")
```

```
                        if int(kan_index) < len(kan_value):
                            break
                    except:
                        pass
                self.jansi.ankan(kan_value[int(kan_index)], self.taku)
                #ドラを表示
                self.taku.show_dora()
                #手牌を表示
                self.show_tehai()
                #もう一度カンや上がれる場合ループする
                if    len(kan_value) > 0 or self.ags.shanten() < 0:
                    continue
            #カンしない(できない)時や和了しない時はループを抜ける
            break

    #リーチするかどうか、その後打牌する
    def do_riichi(self):
        if self.agari_flag == False:
            #リーチしてるならツモ切り
            if self.jansi.get_is_riichi() == True:
                self.jansi.dahai(len(self.jansi.get_tehai())-1)
                #リーチの時の打牌した数をカウント
                self.jansi.riichi_dahai_count += 1
            else:
                if self.ags.shanten() == 0 and self.jansi.naki() == False:
                    #リーチするかどうか
                    while True:
                        try:
                            riichi1 = input("リーチしますか？(y/n): ")
                            if riichi1 == 'y' or riichi1 == 'n':
                                break
                        except:
                            pass
                    if riichi1 == 'y':
                        #リー棒を場に置く
                        self.jansi.sub_tenbou(1000)
                        self.taku.set_oki_riibou()
                        print("リーチで切る牌を選んでください")
                        self.jansi.set_is_riichi()
```

```
                    #ダブルリーチかどうか
                    if self.jansi.dahai_count == 0:
                        self.jansi.is_daburu_riichi = True
                #どの牌を切るか
                self.dahai()

#どの牌を切るかの処理
def dahai(self):
    #牌を切る動作
    while True:
        try:
            value = input("どの牌を切りますか？: ")
            karioki = self.jansi.get_tehai()[int(value)]
            self.jansi.dahai(int(value))
            self.ags.tehai_henkan(self.jansi.get_unit_tehai())
            #リーチが直前にかかっているなら
            if self.jansi.get_is_riichi() == True:
                self.ags.tehai_henkan(self.jansi.get_tehai())
                if self.ags.shanten() > 0:
                    print("その牌は切れません")
                    print("")
                    #再び手牌に仮置きした牌を戻す
                    self.jansi.get_tehai().append(karioki)
                    #理牌
                    self.jansi.riipai()
                    #手牌を表示
                    self.show_tehai()
                    #一度捨てようとした牌を捨て牌に追加しない
                    del self.jansi.get_sutehai()[-1]
                    continue
            #リンシャン等の判定をリセット
            self.jansi.reset_yaku()
            #自分の番の後、他が鳴いたかどうかのフラグをリセット
            self.naki = False
            #打牌した数をカウント
            self.jansi.dahai_count += 1
            break
        except:
            pass
```

```
# 1 局を終わるときの処理
def finish_round(self, dareka_agari):
    nagasi = False
    #誰も上がっていないなら
    if dareka_agari == False:
        self.ags.tehai_henkan(self.jansi.get_tehai())
        #テンパイかどうか
        if self.ags.shanten() <= 0:
            print(self.name + "はテンパイでした")
            #手牌の表示
            self.show_tehai()
            self.tenpai_flag = True
        else:
            print(self.name + "はノーテンでした")
            self.tenpai_flag = False
        #流し満貫かどうか
        nagasi = True
        for i in self.jansi.sutehai:
            if i == 0 or i ==8 or i ==9 or i ==17 or i ==18 or i ==26 or i ==27¥
                    or i ==28 or i ==29 or i ==30 or i ==31 or i ==32 or i ==33:
                pass
            else:
                nagasi = False
        #流し満貫なら,ノーテンでも和了する
        if nagasi == True:
            print(self.name + "は流し満貫でした")
            self.jansi.is_nagashi_mangan = True
            #上がるための流れ(全部必要)
            self.jansi.unit_agari()
            self.ags.tehai_henkan(self.jansi.get_unit_tehai())
            self.move_tenbou = self.calculation()
            #add_tenbou()のために必要
            self.agari_flag = True
            self.add_tenbou2()
            self.ags.print_hand_result()
    #牌をリセットする
    self.jansi.reset_hai()
    #役をリセットする
```

```
        self.jansi.reset_yaku()
        #リーチの役をリセットする
        self.jansi.is_riichi = False
        #自分の番の後、他が鳴いたかどうかのフラグをリセット
        self.naki = False
        #リーチ後に見逃しで振り聴になるフラグ
        self.huriten2 = False
        return nagasi

#ロンの処理
def ron(self, sutehai, name):
        #捨て牌を手牌に加える
        self.jansi.get_tehai().append(sutehai)
        self.ags.tehai_henkan(self.jansi.get_tehai())
        #上がれても役がないとダメ
        how_agari = False
        #上がるかどうかの処理(役があるかどうか)
        if self.ags.shanten() < 0:
                #捨てた牌をツモ牌に設定する
                self.jansi.set_tumohai(sutehai)
                #人和かどうか
                if self.jansi.dahai_count == 0 and self.naki == False¥
                        and self.jansi.ankanhai == []:
                        self.jansi.is_renhou = True
                #一発かどうか
                if self.jansi.is_riichi == True and self.naki == False and¥
                                self.jansi.riichi_dahai_count == 1:
                        self.jansi.is_ippatsu = True
                else:
                        self.jansi.is_ippatsu = False
                #河底かどうか
                if len(self.taku.get_yama()) == 14:
                        self.jansi.is_houtei = True
                #上がるための流れ(全部必要)
                self.jansi.unit_agari()
                self.ags.tehai_henkan(self.jansi.get_unit_tehai())
                try:
                        self.move_tenbou = self.calculation()
                        how_agari = True
```

```
                except:
                    pass
        if how_agari == False:
            #一度手に加えた牌を捨てる
            del self.jansi.get_tehai()[-1]
        elif how_agari == True:
            #振り聴かどうか
            huriten = False
            #上がり牌の検索
            agarihai = self.search_agarihai()
            #もしロンしようとした手が振り聴だったら上がれない
            for j in agarihai:
                for i in self.jansi.get_sutehai():
                    if i == j:
                        huriten = True
            #振り聴じゃない場合
            if huriten == False and self.huriten2 == False:
                while True:
                    try:
                        agaru = input("ロンしますか？  y/n: ")
                        if agaru == 'y' or agaru == 'n':
                            break
                    except:
                        pass
                if agaru == 'y':
                    #上がりの表示
                    self.ags.print_hand_result()
                    self.agari_flag = True
                    self.add_tenbou2()
                    #プレイヤーを表示
                    self.show_name()
                    self.show_tehai()
                    #持ち点を表示
                    self.jansi.show_tenbou()
                    self.agari_flag = True
                elif agaru == 'n':
                    #一度手に加えた牌を捨てる
                    del self.jansi.get_tehai()[-1]
                    #一度見逃したら振り聴
```

```
                if self.jansi.is_riichi == True:
                    self.huriten2 = True
            elif huriten == True:
                print("振り聴です")
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]

#ミンカンの処理
def minkan(self, sutehai):
    if self.jansi.get_is_riichi() == False:
        #捨て牌を手牌に加える
        self.jansi.get_tehai().append(sutehai)
        #牌が 4 つあるものを抽出
        kan_value = [x for x in set (self.jansi.get_tehai())
                            if self.jansi.get_tehai().count(x) > 3]
        #カンするかどうか
        if len(kan_value) > 0 and len(self.taku.get_yama()) > 14:
            kansuru = 'n'
            for i in kan_value:
                if i == sutehai:
                    while True:
                        self.show_tehai()
                        try:
                            kansuru = input("明カンしますか？(y/n): ")
                            if kansuru == 'y' or kansuru == 'n':
                                break
                        except:
                            pass
                    if kansuru == 'y':
                        #明カンする
                        self.jansi.minkan(sutehai, self.taku)
                        #リンシャン牌をツモ牌に設定する(リンシャンのため)
                        self.jansi.rinshanhai = copy.deepcopy(self.jansi.tumohai)
                        #手牌を見せる
                        self.show_tehai()
                        #上がるかカンするか
                        self.agari_kan()
                        #打牌
                        self.dahai()
```

```
                        #ドラを表示
                        self.taku.show_dora()
                        #手牌を表示
                        self.show_tehai()
                        #カンの回数を増やす
                        self.taku.add_kan()
                        #表のドラの数を増やす(明カンなのでリンシャンを得る後に)
                        self.taku.add_dora()
                        return True
                if kansuru == 'n':
                    #一度手に加えた牌を捨てる
                    del self.jansi.get_tehai()[-1]
            else:
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]
        return False

#ポンの処理
def pon(self, sutehai):
    if self.jansi.get_is_riichi() == False:
        #捨て牌を手牌に加える
        self.jansi.get_tehai().append(sutehai)
        #牌が 3 つあるものを抽出
        pon_value = [x for x in set (self.jansi.get_tehai())
                                if self.jansi.get_tehai().count(x) > 2]
        #ポンするかどうか
        if len(pon_value) > 0:
            ponsuru = 'n'
            for i in pon_value:
                if i == sutehai:
                    #一度手に加えた牌を捨てる(表示をわかりやすくするため)
                    del self.jansi.get_tehai()[-1]
                    self.show_tehai()
                    #捨て牌を手牌に加える
                    self.jansi.get_tehai().append(sutehai)
                    while True:
                        try:
                            ponsuru = input("ポンしますか？(y/n): ")
                            if ponsuru == 'y' or ponsuru == 'n':
```

```
                                    break
                            except:
                                pass
                        if ponsuru == 'y':
                            self.jansi.pon(sutehai, self.taku)
                            self.show_tehai()
                            self.dahai()
                            #ドラを表示
                            self.taku.show_dora()
                            #手牌を表示
                            self.show_tehai()
                            return True
                    if ponsuru == 'n':
                        #一度手に加えた牌を捨てる
                        del self.jansi.get_tehai()[-1]
                else:
                    #一度手に加えた牌を捨てる
                    del self.jansi.get_tehai()[-1]
        return False

#チーの処理は一番下にある（長いので）

#カカン
def kakan(self):
        for i in range(len(jansi.ponahi)):
            #ポン牌と同じものを抽出
            kan_value = tehai.index(jansi.ponhai[i][0])
        #カンするかどうか
        if len(kan_value) > 0 and len(taku.get_yama()) > 14:
            kansuru = 'n'
            for i in kan_value:
                while True:
                    self.show_tehai()
                    try:
                        print(Kanji().get_kanji(i),end='')
                        kansuru = input("を加カンしますか？(y/n): ")
                        if kansuru == 'y' or kansuru == 'n':
                            break
                    except:
```

```
                    pass
                if kansuru == 'y':
                    #加カンする
                    self.jansi.kakan(kan_value, self.taku)
                    #リンシャン牌をツモ牌に設定する(リンシャンのため)
                    self.jansi.rinshanhai = copy.deepcopy(self.jansi.tumohai)
                    #手牌を見せる
                    self.show_tehai()
                    #上がるかカンするか
                    self.agari_kan()
                    #打牌
                    self.dahai()
                    #ドラを表示
                    self.taku.show_dora()
                    #手牌を表示
                    self.show_tehai()
                    #カンの回数を増やす
                    self.taku.add_kan()
                    #表のドラの数を増やす(明カンなのでリンシャンを得る後に)
                    self.taku.add_dora()
                    return True
            if kansuru == 'n':
                #一度手に加えた牌を捨てる
                del self.jansi.get_tehai()[-1]
        else:
            #一度手に加えた牌を捨てる
            del self.jansi.get_tehai()[-1]
        return False

    #捨て牌のゲッター
    def get_sutehai(self):
        return self.jansi.get_sutehai()

    #立直のゲッター
    def get_is_riichi(self):
        return self.jansi.is_riichi

    #自風のゲッター
    def get_player_wind(self):
```

```
        return self.jansi.get_player_wind()

    #やり取りする点棒のゲッター
    def get_move_tenbou(self):
        return self.move_tenbou

    #点棒を加える(テンパイ時)
    def add_tenbou(self, tenbou):
        self.jansi.add_tenbou(tenbou)

    #点棒を加える
    def add_tenbou2(self):
        tenbou2 = self.move_tenbou[0] + self.move_tenbou[1]*2 + self.taku.get_honba()*300
        #場のリーチ棒も持っていく
        self.jansi.add_tenbou(tenbou2 + self.taku.get_oki_riibou())

    #点棒を減らす
    def sub_tenbou(self, tenbou2):
        self.jansi.sub_tenbou(tenbou2)

    #点棒
    def get_tenbou(self):
        return self.jansi.tenbou

    #テンパイしてたかどうかのフラグのゲッター
    def get_tenpai_flag(self):
        return self.tenpai_flag

    #上がるためのフラグのゲッター
    def get_agari_flag(self):
        return self.agari_flag

    #上がりフラグのリセット
    def reset_agari_flag(self):
        self.agari_flag = False

    #点棒の表示
    def show_tenbou(self):
        self.jansi.show_tenbou()
```

```
#点棒の表示
def show_tenbou(self):
    self.jansi.show_tenbou()

#自風牌の表示
def show_player_wind(self):
    self.jansi.show_player_wind()

#自風牌を変える
def change_player_wind(self):
    self.jansi.change_player_wind()

#名前の表示
def show_name(self):
    print(" [" + self.name + "] ",end='')

#自風のセッター
def set_player_wind(self, player_wind):
    self.jansi.set_player_wind(player_wind)

#手牌に牌を加える
def append_hai(self, hai):
    self.jansi.get_tehai().append(hai)

#配牌
def haipai(self):
    self.jansi.haipai(self.taku.get_yama())

#理牌
def riipai(self):
    self.jansi.riipai()

#点棒の計算
def calculation(self):
    return self.ags.agari(self.jansi.get_tumohai(),
                          self.taku.get_dora(),
                          self.jansi.get_is_tsumo(),
                          self.jansi.get_is_riichi(),
```

```
                    self.jansi.get_is_ippatsu(),
                    self.jansi.get_is_rinshan(),
                    self.jansi.get_is_chankan(),
                    self.jansi.get_is_haitei(),
                    self.jansi.get_is_houtei(),
                    self.jansi.get_is_daburu_riichi(),
                    self.jansi.get_is_nagashi_mangan(),
                    self.jansi.get_is_tenhou(),
                    self.jansi.get_is_renhou(),
                    self.jansi.get_is_chiihou(),
                    self.jansi.get_player_wind(),
                    self.taku.get_round_wind(),
                    self.jansi.get_ankanhai(),
                    self.jansi.get_minkanhai(),
                    self.jansi.get_ponhai(),
                    self.jansi.get_chiihai())

#手牌を漢字に変換して見せる
def show_tehai(self):
    bangou = ["⓪","①","②","③","④","⑤","⑥",
                "⑦","⑧","⑨","⑩","⑪","⑫","⑬"]
    for i in range(len(self.jansi.get_tehai())):
        print(" " + str(bangou[i]) + "    ",end='')
    print("")
    for i in self.jansi.get_tehai():
        print(Kanji().get_kanji(i),end='')
    print("")

#ツモ牌を漢字に変換して見せる
def show_tumo(self):
    try:
        if self.tehai[13] is not None:
            print("ツモ: " + self.kanji.get_kanji(self.tehai[13]))
    except:
        pass

#チーの処理
def chii(self, sutehai):
    if self.jansi.get_is_riichi() == False:
```

```
#順子を入れる
chii_value = []
#萬子
if sutehai == 0:
    try:
        if self.jansi.get_tehai().index(1) is not None¥
            and self.jansi.get_tehai().index(2) is not None:
            chii_value.append([0,1,2])
    except:
        pass
elif sutehai == 1:
    try:
        if self.jansi.get_tehai().index(0) is not None¥
            and self.jansi.get_tehai().index(2) is not None:
            chii_value.append([0,1,2])
    except:
        pass
    try:
        if self.jansi.get_tehai().index(2) is not None¥
            and self.jansi.get_tehai().index(3) is not None:
            chii_value.append([1,2,3])
    except:
        pass
elif sutehai == 2:
    try:
        if self.jansi.get_tehai().index(0) is not None¥
            and self.jansi.get_tehai().index(1) is not None:
            chii_value.append([0,1,2])
    except:
        pass
    try:
        if self.jansi.get_tehai().index(1) is not None¥
            and self.jansi.get_tehai().index(3) is not None:
            chii_value.append([1,2,3])
    except:
        pass
    try:
        if self.jansi.get_tehai().index(3) is not None¥
            and self.jansi.get_tehai().index(4) is not None:
```

```
                chii_value.append([2,3,4])
        except:
            pass
    elif sutehai == 3:
        try:
            if self.jansi.get_tehai().index(1) is not None¥
                and self.jansi.get_tehai().index(2) is not None:
                chii_value.append([1,2,3])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(2) is not None¥
                and self.jansi.get_tehai().index(4) is not None:
                chii_value.append([2,3,4])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(4) is not None¥
                and self.jansi.get_tehai().index(5) is not None:
                chii_value.append([3,4,5])
        except:
            pass
    elif sutehai == 4:
        try:
            if self.jansi.get_tehai().index(2) is not None¥
                and self.jansi.get_tehai().index(3) is not None:
                chii_value.append([2,3,4])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(3) is not None¥
                and self.jansi.get_tehai().index(5) is not None:
                chii_value.append([3,4,5])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(5) is not None¥
                and self.jansi.get_tehai().index(6) is not None:
                chii_value.append([4,5,6])
```

```
        except:
            pass
    elif sutehai == 5:
        try:
            if self.jansi.get_tehai().index(3) is not None¥
                and self.jansi.get_tehai().index(4) is not None:
                chii_value.append([3,4,5])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(4) is not None¥
                and self.jansi.get_tehai().index(6) is not None:
                chii_value.append([4,5,6])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(6) is not None¥
                and self.jansi.get_tehai().index(7) is not None:
                chii_value.append([5,6,7])
        except:
            pass
    elif sutehai == 6:
        try:
            if self.jansi.get_tehai().index(4) is not None¥
                and self.jansi.get_tehai().index(5) is not None:
                chii_value.append([4,5,6])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(5) is not None¥
                and self.jansi.get_tehai().index(7) is not None:
                chii_value.append([5,6,7])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(7) is not None¥
                and self.jansi.get_tehai().index(8) is not None:
                chii_value.append([6,7,8])
        except:
```

```
            pass
    elif sutehai == 7:
        try:
            if self.jansi.get_tehai().index(5) is not None¥
                and self.jansi.get_tehai().index(6) is not None:
                chii_value.append([5,6,7])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(6) is not None¥
                and self.jansi.get_tehai().index(8) is not None:
                chii_value.append([6,7,8])
        except:
            pass
    elif sutehai == 8:
        try:
            if self.jansi.get_tehai().index(6) is not None¥
                and self.jansi.get_tehai().index(7) is not None:
                chii_value.append([6,7,8])
        except:
            pass
    #筒子
    elif sutehai == 9:
        try:
            if self.jansi.get_tehai().index(10) is not None¥
                and self.jansi.get_tehai().index(11) is not None:
                chii_value.append([9,10,11])
        except:
            pass
    elif sutehai == 10:
        try:
            if self.jansi.get_tehai().index(9) is not None¥
                and self.jansi.get_tehai().index(11) is not None:
                chii_value.append([9,10,11])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(11) is not None¥
                and self.jansi.get_tehai().index(12) is not None:
```

```
                chii_value.append([10,11,12])
        except:
            pass
    elif sutehai == 11:
        try:
            if self.jansi.get_tehai().index(9) is not None¥
                and self.jansi.get_tehai().index(10) is not None:
                chii_value.append([9,10,11])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(10) is not None¥
                and self.jansi.get_tehai().index(12) is not None:
                chii_value.append([10,11,12])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(12) is not None¥
                and self.jansi.get_tehai().index(13) is not None:
                chii_value.append([11,12,13])
        except:
            pass
    elif sutehai == 12:
        try:
            if self.jansi.get_tehai().index(10) is not None¥
                and self.jansi.get_tehai().index(11) is not None:
                chii_value.append([10,11,12])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(11) is not None¥
                and self.jansi.get_tehai().index(13) is not None:
                chii_value.append([11,12,13])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(13) is not None¥
                and self.jansi.get_tehai().index(14) is not None:
                chii_value.append([12,13,14])
```

```
        except:
            pass
    elif sutehai == 13:
        try:
            if self.jansi.get_tehai().index(11) is not None¥
                and self.jansi.get_tehai().index(12) is not None:
                chii_value.append([11,12,13])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(12) is not None¥
                and self.jansi.get_tehai().index(14) is not None:
                chii_value.append([12,13,14])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(14) is not None¥
                and self.jansi.get_tehai().index(15) is not None:
                chii_value.append([13,14,15])
        except:
            pass
    elif sutehai == 14:
        try:
            if self.jansi.get_tehai().index(12) is not None¥
                and self.jansi.get_tehai().index(13) is not None:
                chii_value.append([12,13,14])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(13) is not None¥
                and self.jansi.get_tehai().index(15) is not None:
                chii_value.append([13,14,15])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(15) is not None¥
                and self.jansi.get_tehai().index(16) is not None:
                chii_value.append([14,15,16])
        except:
```

```
            pass
    elif sutehai == 15:
        try:
            if self.jansi.get_tehai().index(13) is not None¥
                and self.jansi.get_tehai().index(14) is not None:
                chii_value.append([13,14,15])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(14) is not None¥
                and self.jansi.get_tehai().index(16) is not None:
                chii_value.append([14,15,16])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(16) is not None¥
                and self.jansi.get_tehai().index(17) is not None:
                chii_value.append([15,16,17])
        except:
            pass
    elif sutehai == 16:
        try:
            if self.jansi.get_tehai().index(14) is not None¥
                and self.jansi.get_tehai().index(15) is not None:
                chii_value.append([14,15,16])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(15) is not None¥
                and self.jansi.get_tehai().index(17) is not None:
                chii_value.append([15,16,17])
        except:
            pass
    elif sutehai == 17:
        try:
            if self.jansi.get_tehai().index(15) is not None¥
                and self.jansi.get_tehai().index(16) is not None:
                chii_value.append([15,16,17])
        except:
```

```
            pass
    #索子
    elif sutehai == 18:
        try:
            if self.jansi.get_tehai().index(19) is not None¥
                and self.jansi.get_tehai().index(20) is not None:
                chii_value.append([18,19,20])
        except:
            pass
    elif sutehai == 19:
        try:
            if self.jansi.get_tehai().index(18) is not None¥
                and self.jansi.get_tehai().index(20) is not None:
                chii_value.append([18,19,20])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(20) is not None¥
                and self.jansi.get_tehai().index(21) is not None:
                chii_value.append([19,20,21])
        except:
            pass
    elif sutehai == 20:
        try:
            if self.jansi.get_tehai().index(18) is not None¥
                and self.jansi.get_tehai().index(19) is not None:
                chii_value.append([18,19,20])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(19) is not None¥
                and self.jansi.get_tehai().index(21) is not None:
                chii_value.append([19,20,21])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(21) is not None¥
                and self.jansi.get_tehai().index(22) is not None:
                chii_value.append([20,21,22])
```

```
        except:
            pass
    elif sutehai == 21:
        try:
            if self.jansi.get_tehai().index(19) is not None¥
                and self.jansi.get_tehai().index(20) is not None:
                chii_value.append([19,20,21])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(20) is not None¥
                and self.jansi.get_tehai().index(22) is not None:
                chii_value.append([20,21,22])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(22) is not None¥
                and self.jansi.get_tehai().index(23) is not None:
                chii_value.append([21,22,23])
        except:
            pass
    elif sutehai == 22:
        try:
            if self.jansi.get_tehai().index(20) is not None¥
                and self.jansi.get_tehai().index(21) is not None:
                chii_value.append([20,21,22])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(21) is not None¥
                and self.jansi.get_tehai().index(23) is not None:
                chii_value.append([21,22,23])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(23) is not None¥
                and self.jansi.get_tehai().index(24) is not None:
                chii_value.append([22,23,24])
        except:
```

```
            pass
    elif sutehai == 23:
        try:
            if self.jansi.get_tehai().index(21) is not None¥
                and self.jansi.get_tehai().index(22) is not None:
                chii_value.append([21,22,23])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(22) is not None¥
                and self.jansi.get_tehai().index(24) is not None:
                chii_value.append([22,23,24])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(24) is not None¥
                and self.jansi.get_tehai().index(25) is not None:
                chii_value.append([23,24,25])
        except:
            pass
    elif sutehai == 24:
        try:
            if self.jansi.get_tehai().index(22) is not None¥
                and self.jansi.get_tehai().index(23) is not None:
                chii_value.append([22,23,24])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(23) is not None¥
                and self.jansi.get_tehai().index(25) is not None:
                chii_value.append([23,24,25])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(25) is not None¥
                and self.jansi.get_tehai().index(26) is not None:
                chii_value.append([24,25,26])
        except:
            pass
```

```
elif sutehai == 25:
        try:
            if self.jansi.get_tehai().index(23) is not None¥
                and self.jansi.get_tehai().index(24) is not None:
                chii_value.append([23,24,25])
        except:
            pass
        try:
            if self.jansi.get_tehai().index(24) is not None¥
                and self.jansi.get_tehai().index(26) is not None:
                chii_value.append([24,25,26])
        except:
            pass
elif sutehai == 26:
        try:
            if self.jansi.get_tehai().index(24) is not None¥
                and self.jansi.get_tehai().index(25) is not None:
                chii_value.append([24,25,26])
        except:
            pass
#チーするかどうか
if len(chii_value) > 0:
        chiisuru = 'a'
        #理牌する
        self.jansi.riipai()
        self.show_tehai()
        while True:
            try:
                chiisuru = input("チーしますか？(y/n): ")
                if chiisuru == 'y' or chiisuru == 'n':
                    break
            except:
                pass
        if chiisuru == 'y':
            #チーできる牌の最初の組み合わせを選ぶ
            dore = 0
            #２個以上チーができるなら
            if len(chii_value) > 1:
                while True:
```

```
                                chii01 = []
                                for i in chii_value:
                                    chii00 = []
                                    for j in range(3):
                                        chii00.append(Kanji().get_kanji(i[j]))
                                    chii01.append(chii00)
                                print(chii01)
                                print("どのチーをしますか？(左から 0,1・・・)",end='')
                                dore = input("：  ")
                                if int(dore) < len(chii_value):
                                    break
                        self.jansi.get_tehai().append(sutehai)
                        self.jansi.chii(chii_value[int(dore)], self.taku)
                        self.show_tehai()
                        self.dahai()
                        #捨て牌を回収する
                        del sutehai
                        return True
        return False

#なき牌を返す
def unit_nakihai(self):
        return self.jansi.unit_nakihai()

#上がり牌の検索
def search_agarihai(self):
        #仮りの上がりの検索
        ags2 = Agari_search()
        agari_list = []
        #上がり牌を探す
        for i in range(34):
            #仮りの手牌
            tehai2 = copy.deepcopy(self.jansi.get_tehai())
            del tehai2[-1]
            #仮りの手牌に一つ牌を入れてみる
            tehai2.append(i)
            ags2.tehai_henkan(tehai2)
            #上がり牌ならリストに入れる
            if ags2.shanten() < 0:
```

```
                agari_list.append(i)
        #上がり牌を返す
        return agari_list
```

<Jansi.py>

```
# -*- coding: utf-8 -*-
from Taku import Taku
from Kanji import Kanji
#風(場&自)
from mahjong.constants import EAST, SOUTH, WEST, NORTH
import copy

#一人の雀士としての挙動
class Jansi(object):

    def __init__(self):
        #手牌
        self.tehai = []
        #捨て牌
        self.sutehai = []
        #ツモ牌
        self.tumohai = 0
        #打牌した数を数える(テンホーダブリー用)
        self.dahai_count = 0
        #リーチ後に打牌した数を数える(一発用)
        self.riichi_dahai_count = 0
        #暗カンした牌
        self.ankanhai = []
        #明カンした牌
        self.minkanhai = []
        #ポンした牌
        self.ponhai = []
        #チーした牌
        self.chiihai = []
        #嶺上牌
        self.rinshanhai = -1
        #加カン牌
        self.chankanhai = -1
        #上がるために仮置きする手牌
```

```
        self.unit_tehai = []
        #自風
        self.player_wind = EAST
        #漢字に変換するインスタンスの作成
        self.kanji = Kanji()
        #点数
        self.tenbou = 25000
        #リーチ牌を入れる
        self.riichi_hai = -1
        #役に必要なもの
        self.is_tsumo = False
        self.is_riichi = False
        self.is_ippatsu = False
        self.is_rinshan = False
        self.is_chankan = False
        self.is_haitei = False
        self.is_houtei = False
        self.is_daburu_riichi = False
        self.is_nagashi_mangan = False
        self.is_tenhou = False
        self.is_renhou = False
        self.is_chiihou = False

    #配牌を得る
    def haipai(self, yama):
        self.tehai =    yama[0:13]
        del yama[0:13]
        return self.tehai

    #手牌を見やすくする(理牌)
    def riipai(self):
        self.tehai.sort()

    #ツモを得る
    def tumo(self, yama):
        self.tumohai = yama[0]
        del yama[0]
        self.tehai.append(self.tumohai)
        return self.tumohai
```

```
#決められた牌を切る
def dahai(self,value):
    # ここで打牌選択のロジック
    hai = self.tehai[value]
    del self.tehai[value]
    self.sutehai.append(hai)
    self.dahai_count += 1
    return hai

#暗カン(関数 value は同じ数字の値)
def ankan(self, value, taku):
    #理牌する
    self.riipai()
    #手牌から value に当てはまる牌の位置を覚える配列
    index = [i for i, x in enumerate(self.tehai) if x == value]
    #カンした牌を仮に入れる
    beta = []
    #カンした牌を手牌から除く
    for i in index:
        beta.append(self.tehai[i])
    for i in index:
        del self.tehai[index[0]]
    #カンした牌を横にやる
    self.ankanhai.append(beta)
    #カンの回数を増やす
    taku.add_kan()
    #表のドラの数を増やす(暗カンなのでリンシャンを得る前に)
    taku.add_dora()
    #リンシャンを得る
    self.tumohai = taku.get_yama()[-1]
    #リンシャン牌をツモ牌に設定する(リンシャンのため)
    self.rinshanhai = copy.deepcopy(self.tumohai)
    del taku.get_yama()[-1]
    self.tehai.append(self.tumohai)

#明カン(関数 value は同じ数字の値)(ここでドラはめくらない)
def minkan(self, value, taku):
    #理牌する
```

```
        self.riipai()
        #手牌から value に当てはまる牌の位置を覚える配列
        index = [i for i, x in enumerate(self.tehai) if x == value]
        #カンした牌を仮に入れる
        beta = []
        for i in index:
            beta.append(self.tehai[i])
        for i in index:
            del self.tehai[index[0]]
        #カンした牌を横にやる
        self.minkanhai.append(beta)
        #リンシャンを得る
        self.tumohai = taku.get_yama()[-1]
        del taku.get_yama()[-1]
        self.tehai.append(self.tumohai)

#ポン
def pon(self, value):
        #理牌する
        self.riipai()
        #手牌から value に当てはまる牌の位置を覚える配列
        index = [i for i, x in enumerate(self.tehai) if x == value]
        #カンした牌を仮に入れる
        beta = []
        for i in index:
            beta.append(self.tehai[i])
        for i in index:
            del self.tehai[index[0]]
        #ポンした牌を横にやる
        self.ponhai.append(beta)

#チー(この value は配列(連なった 3 つの数字を入れる))
def chii(self, value):
        #リー牌
        self.riipai()
        #チーした牌を仮に入れる
        beta = []
        #手牌から value に当てはまる牌の位置を覚える配列
        index1 = [i for i, x in enumerate(self.tehai) if x == value[0]]
```

```
        beta.append(self.tehai[index1[0]])
        del self.tehai[index1[0]]
        index2 = [i for i, x in enumerate(self.tehai) if x == value[1]]
        beta.append(self.tehai[index2[0]])
        del self.tehai[index2[0]]
        index3 = [i for i, x in enumerate(self.tehai) if x == value[2]]
        beta.append(self.tehai[index3[0]])
        del self.tehai[index3[0]]
        #チーした牌を横にやる
        self.chiihai.append(beta)

    #加カン
    def kakan(self, value, taku):
        #理牌する
        self.riipai()
        #手牌から value に当てはまる牌の位置を覚える配列
        index = [i for i, x in enumerate(self.tehai) if x == value]
        #ポンした牌を消すための処理
        count = -1
        for i in self.ponhai:
            count += 1
            if i[0] == value:
                break
        #カンした牌を仮に入れる
        beta = []
        for i in range(4):
            beta.append(self.tehai[index[0]])
        #手牌とポン牌からカンした牌を消す
        del self.tehai[index[0]]
        del self.ponhai[count]
        #カンした牌を横にやる
        self.minkanhai.append(beta)
        self.chankanhai = value
        #リンシャンを得る
        self.tumohai = taku.get_yama()[-1]
        del taku.get_yama()[-1]
        self.tehai.append(self.tumohai)
```

```
#捨て牌を漢字に変換して見せる
def show_sutehai(self):
    print("捨て牌:",end='')
    if self.is_riichi == True and self.riichi_dahai_count == 0:
        self.riichi_hai = len(self.sutehai)-1
    for i in range(len(self.sutehai)):
        if i == self.riichi_hai:
            print("#",end='')
        print(str(self.kanji.get_kanji(self.sutehai[i])),end='')
        if (i+1) % 6 != 0 :
            print("・",end='')
        else:
            print("")
            print("　　",end='')
    print("")
    print("")

#鳴いた牌を表示する
def show_nakihai(self):
    print("鳴き牌")
    if len(self.ankanhai) > 0:
        print("[暗カン: " ,end = '')
        for i in self.ankanhai:
            print("[", end='')
            for j in range(4):
                print(Kanji().get_kanji(i[j]), end = '')
            print("]", end='')
        print("]")
    if len(self.minkanhai) > 0:
        print("[明カン: " ,end = '')
        for i in self.minkanhai:
            print("[", end='')
            for j in range(4):
                print(Kanji().get_kanji(i[j]), end = '')
            print("]", end='')
        print("]")
    if len(self.ponhai) > 0:
        print("[ポン: " ,end = '')
        for i in self.ponhai:
```

```
                print("[", end='')
                for j in range(3):
                    print(Kanji().get_kanji(i[j]), end = '')
                print("]", end='')
            print("]")
        if len(self.chiihai) > 0:
            print("[チー: " ,end = '')
            for i in self.chiihai:
                print("[", end='')
                for j in range(3):
                    print(Kanji().get_kanji(i[j]), end = '')
                print("]", end='')
            print("]")

#点数の表示
def show_tenbou(self):
    print(str(int(self.tenbou)) + "点", end='')

#自風を表示
def show_player_wind(self):
    wind = ''
    if self.player_wind == EAST:
        wind ='東'
    elif self.player_wind == SOUTH:
        wind ='南'
    elif self.player_wind == WEST:
        wind ='西'
    elif self.player_wind == NORTH:
        wind ='北'
    print("[" + wind + "]" , end='')


#自風を変える
def change_player_wind(self):
    if self.player_wind == EAST:
        self.player_wind = NORTH
    elif self.player_wind == SOUTH:
        self.player_wind = EAST
    elif self.player_wind == WEST:
```

```
            self.player_wind = SOUTH
        elif self.player_wind == NORTH:
            self.player_wind = WEST

#点棒を加える
def add_tenbou(self, tensuu):
        self.tenbou += tensuu

#点棒を減らす
def sub_tenbou(self, tensuu):
        self.tenbou -= tensuu

#鳴いた牌と手牌を結合する(和了を計算するため)
def unit_agari(self):
        self.unit_tehai = copy.deepcopy(self.tehai)
        if len(self.ankanhai) > 0:
            for i in range(len(self.ankanhai)):
                for j in range(3):
                    self.unit_tehai.append(self.ankanhai[i][j])
        if len(self.minkanhai) > 0:
            for i in range(len(self.minkanhai)):
                for j in range(3):
                    self.unit_tehai.append(self.minkanhai[i][j])
        if len(self.ponhai) > 0:
            for i in range(len(self.ponhai)):
                for j in range(3):
                    self.unit_tehai.append(self.ponhai[i][j])
        if len(self.chiihai) > 0:
            for i in range(len(self.chiihai)):
                for j in range(3):
                    self.unit_tehai.append(self.chiihai[i][j])

#鳴いた牌を結合する(卓上の鳴いた牌の数を確認するため、AI で使用)
def unit_nakihai(self):
        nakihai = []
        if len(self.ankanhai) > 0:
            for i in range(len(self.ankanhai)):
                for j in range(4):
                    nakihai.append(self.ankanhai[i][j])
```

```
        if len(self.minkanhai) > 0:
            for i in range(len(self.minkanhai)):
                for j in range(4):
                    nakihai.append(self.minkanhai[i][j])
        if len(self.ponhai) > 0:
            for i in range(len(self.ponhai)):
                for j in range(3):
                    nakihai.append(self.ponhai[i][j])
        if len(self.chiihai) > 0:
            for i in range(len(self.chiihai)):
                for j in range(3):
                    nakihai.append(self.chiihai[i][j])
        return nakihai

#手牌のゲッター
def get_tehai(self):
        return self.tehai

#上がるための手牌の仮置きのゲッター
def get_unit_tehai(self):
        return self.unit_tehai

#手牌のセッター
def set_tehai(self, tehai):
        self.tehai = tehai

#捨牌のゲッター
def get_sutehai(self):
        return self.sutehai

#ツモ牌のセッター(ロン専用)
def set_tumohai(self, tumohai):
        self.tumohai = tumohai

#ツモ牌のゲッター
def get_tumohai(self):
        return self.tumohai

#自風のゲッター
```

```
def get_player_wind(self):
    return self.player_wind

#自風のセッター
def set_player_wind(self, player_wind):
    self.player_wind = player_wind

#暗カンのゲッター
def get_ankanhai(self):
    return self.ankanhai

#明カンのゲッター
def get_minkanhai(self):
    return self.minkanhai

#ポンのゲッター
def get_ponhai(self):
    return self.ponhai

#チーのゲッター
def get_chiihai(self):
    return self.chiihai

#鳴いているかどうか
def naki(self):
    #鳴いているなら True
    if len(self.minkanhai) > 0 or len(self.ponhai) > 0 or len(self.chiihai) > 0:
        return True
    else:
        return False

#ツモしたかどうかのゲッター
def get_is_tsumo(self):
    return self.is_tsumo

#ツモしたかどうかのセッター
def set_is_tsumo(self, tsumo):
    self.is_tsumo = tsumo
```

```
#リーチしたかどうかのゲッター
def get_is_riichi(self):
    return self.is_riichi
#リーチしたかどうかのセッター
def set_is_riichi(self):
    self.is_riichi = True

#一発したかどうかのゲッター
def get_is_ippatsu(self):
    return self.is_ippatsu
#一発したかどうかのセッター
def set_is_ippatsu(self):
    self.is_ippatsu = True

#嶺上開花したかどうかのゲッター
def get_is_rinshan(self):
    return self.is_rinshan
#嶺上開花したかどうかのセッター
def set_is_rinshan(self, rinshan):
    self.is_rinshan = rinshan

#チャンカンしたかどうかのゲッター
def get_is_chankan(self):
    return self.is_chankan
#チャンカンしたかどうかのセッター
def set_is_chankan(self):
    self.is_chankan = True

#ハイテイしたかどうかのゲッター
def get_is_haitei(self):
    return self.is_haitei
#ハイテイしたかどうかのセッター
def set_is_haitei(self):
    self.is_haitei = True

#ホウテイしたかどうかのゲッター
def get_is_houtei(self):
    return self.is_houtei
#ホウテイしたかどうかのセッター
```

```
def set_is_houtei(self):
    self.is_houtei = True

#ダブルリーチしたかどうかのゲッター
def get_is_daburu_riichi(self):
    return self.is_daburu_riichi
#ダブルリーチしたかどうかのセッター
def set_is_daburu_riichi(self):
    self.is_daburu_riichi = True

#流し満貫したかどうかのゲッター
def get_is_nagashi_mangan(self):
    return self.is_nagashi_mangan
#流し満貫したかどうかのセッター
def set_is_nagashi_mangan(self):
    self.is_nagashi_mangan = True

#天和したかどうかのゲッター
def get_is_tenhou(self):
    return self.is_tenhou
#天和したかどうかのセッター
def set_is_tenhou(self):
    self.is_tenhou = True

#人和したかどうかのゲッター
def get_is_renhou(self):
    return self.is_renhou
#人和したかどうかのセッター
def set_is_renhou(self):
    self.is_renhou = True

#地和したかどうかのゲッター
def get_is_chiihou(self):
    return self.is_chiihou
#地和したかどうかのセッター
def set_is_chiihou(self):
    self.is_chiihou = True

#役に必要なものを全て False に戻す
```

```
    def reset_yaku(self):
        self.is_tsumo = False
        self.is_ippatsu = False
        self.is_rinshan = False
        self.is_chankan = False
        self.is_haitei = False
        self.is_houtei = False
        self.is_daburu_riichi = False
        self.is_nagashi_mangan = False
        self.is_tenhou = False
        self.is_renhou = False
        self.is_chiihou = False
        self.dahai_count = 0
        self.riichi_dahai_count = 0
        self.riichi_hai = -1

    #全ての牌情報をリセットする
    def reset_hai(self):
        self.sutehai = []
        self.tehai = []
        self.ankanhai = []
        self.minkanhai = []
        self.ponhai = []
        self.chiihai = []
```

<Kanji.py>

```
# -*- coding: utf-8 -*-
class Kanji(object):

    def get_kanji(self, value):
        if value == 0:
            return "1 萬 "
        elif value == 1:
            return "2 萬 "
        elif value == 2:
            return "3 萬 "
        elif value == 3:
            return "4 萬 "
        elif value == 4:
```

```
        return "5 萬  "
elif value == 5:
        return "6 萬  "
elif value == 6:
        return "7 萬  "
elif value == 7:
        return "8 萬  "
elif value == 8:
        return "9 萬  "
elif value == 9:
        return "1◯    "
elif value == 10:
        return "2◯    "
elif value == 11:
        return "3◯    "
elif value == 12:
        return "4◯    "
elif value == 13:
        return "5◯    "
elif value == 14:
        return "6◯    "
elif value == 15:
        return "7◯    "
elif value == 16:
        return "8◯    "
elif value == 17:
        return "9◯    "
elif value == 18:
        return "1 s   "
elif value == 19:
        return "2 s   "
elif value == 20:
        return "3 s   "
elif value == 21:
        return "4 s   "
elif value == 22:
        return "5 s   "
elif value == 23:
        return "6 s   "
```

```
        elif value == 24:
            return "7ｓ　"
        elif value == 25:
            return "8ｓ　"
        elif value == 26:
            return "9ｓ　"
        elif value == 27:
            return "　東　"
        elif value == 28:
            return "　南　"
        elif value == 29:
            return "　西　"
        elif value == 30:
            return "　北　"
        elif value == 31:
            return "　白　"
        elif value == 32:
            return "　發　"
        elif value == 33:
            return "　中　"
```

<Taku.py>

```
# -*- coding: utf-8 -*-
from Kanji import Kanji
import random
#風(場&自)
from mahjong.constants import EAST, SOUTH, WEST, NORTH
import copy

class Taku(object):

    def __init__(self):
        #牌全種
        self.yama = []
        #場風
        self.round_wind = EAST
        #はじめに親だった人(順位を決める時に使用)
        self.start_East = None
        #局数
```

```
        self.round = 0
        #本場
        self.honba = 0
        #ドラ
        self.dora = []
        #カンの数
        self.kan = 0
        #流れた時に置くリー棒
        self.oki_riibou = 0
        #捨て牌
        self.jansi1_sutehai = []
        self.jansi2_sutehai = []
        self.jansi3_sutehai = []
        self.jansi4_sutehai = []
        #鳴き牌
        self.jansi1_nakihai = []
        self.jansi2_nakihai = []
        self.jansi3_nakihai = []
        self.jansi4_nakihai = []
        #リーチしてるかどうか
        self.jansi1_riichi = False
        self.jansi2_riichi = False
        self.jansi3_riichi = False
        self.jansi4_riichi = False
        #安全パイ(捨て牌+リーチした後の他の人の打牌)
        self.jansi1_anzenpai = []
        self.jansi2_anzenpai = []
        self.jansi3_anzenpai = []
        self.jansi4_anzenpai = []

    #場風を変える
    def change_round_wind(self):
        if self.round_wind == EAST:
            self.round_wind = SOUTH
        elif self.round_wind == SOUTH:
            self.round_wind = WEST
        elif self.round_wind == WEST:
            self.round_wind = NORTH
        elif self.round_wind == NORTH:
```

```
            self.round_wind = EAST

    #局数を増やす
    def add_round(self):
        self.round += 1

    # 1 本場増やす
    def add_honba(self):
        self.honba += 1

    #カンの回数を増やす
    def add_kan(self):
        self.kan += 1

    #表になってるドラを増やす(リーチがかかってない時)
    def add_dora(self):
        #初めのドラ
        if self.kan == 0:
            self.dora.append(self.yama[-6])
        #カンドラ 1
        elif self.kan == 1:
            self.dora.append(self.yama[-8])
        #カンドラ 2
        elif self.kan == 2:
            self.dora.append(self.yama[-10])
        #カンドラ 3
        elif self.kan == 3:
            self.dora.append(self.yama[-12])
        #カンドラ 4
        elif self.kan == 4:
            self.dora.append(self.yama[-14])

    #リーチがかかっているなら裏ドラを乗せる
    def add_uradora(self):
        #裏ドラ 1
        if self.kan >= 0:
            self.dora.append(self.yama[-5])
        #裏ドラ 2
        if self.kan >= 1:
```

```
            self.dora.append(self.yama[-7])
        #裏ドラ 3
        if self.kan >= 2:
            self.dora.append(self.yama[-9])
        #裏ドラ 4
        if self.kan >= 3:
            self.dora.append(self.yama[-11])
        #裏ドラ 5
        if self.kan >= 4:
            self.dora.append(self.yama[-13])

#リンシャン牌を得る
def get_rinshan(self):
        rinshan = 0
        if self.kan == 1:
            rinshan = self.yama[-1]
            del self.yama[-1]
        elif self.kan == 2:
            rinshan = self.yama[-2]
            del self.yama[-2]
        elif self.kan == 3:
            rinshan = self.yama[-3]
            del self.yama[-3]
        elif self.kan == 4:
            rinshan = self.yama[-4]
            del self.yama[-4]
        return rinshan

def yama_reset(self):
        #牌全種
        self.yama =    [0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3,
                            4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,
                            8,8,8,8,9,9,9,9,10,10,10,10,11,11,11,11,
                            12,12,12,12,13,13,13,13,14,14,14,14,15,15,15,15,
                            16,16,16,16,17,17,17,17,18,18,18,18,19,19,19,19,
                            20,20,20,20,21,21,21,21,22,22,22,22,23,23,23,23,
                            24,24,24,24,25,25,25,25,26,26,26,26,27,27,27,27,
                            28,28,28,28,29,29,29,29,30,30,30,30,31,31,31,31,
                            32,32,32,32,33,33,33,33]
```

```
        random.shuffle(self.yama)

    #局数をリセットする
    def reset_round(self):
        self.round = 0

    #本場をリセットする
    def reset_honba(self):
        self.honba = 0

    #ドラをリセットする
    def reset_dora(self):
        self.dora = []
        self.kan = 0

    #局数を表示
    def show_round(self):
        wind = ''
        if self.round_wind == EAST:
            wind ='東'
        elif self.round_wind == SOUTH:
            wind ='南'
        elif self.round_wind == WEST:
            wind ='西'
        elif self.round_wind == NORTH:
            wind ='北'
        print(wind + str(self.round + 1) + "局　・",end='')

    #本場を表示
    def show_honba(self):
        print(str(self.honba) + "本場")

    #ドラを表示
    def show_dora(self):
        print("ドラ表示牌[ ",end='')
        value = len(self.dora)
        for dr in self.dora:
            print(str(Kanji().get_kanji(dr)),end='')
            value -= 1
```

```
            if value != 0:
                print(",",end='')
        print("]")

#山の数の表示
def show_yama(self):
        print("残り牌: ",end = '')
        print(len(self.yama)-1)

#場風のゲッター
def get_round_wind(self):
        return self.round_wind

#局数のゲッター
def get_round(self):
        return self.round

#局数のセッター
def set_round(self, round):
        self.round = round

#ドラのゲッター
def get_dora(self):
        return self.dora

#本場のゲッター
def get_honba(self):
        return self.honba

#山のゲッター
def get_yama(self):
        return self.yama

#リーチ棒を場に置く
def set_oki_riibou(self):
        self.oki_riibou += 1000

#リーチ棒を回収する
def get_oki_riibou(self):
```

```
        oki_riibou2 = copy.deepcopy(self.oki_riibou)
        self.oki_riibou = 0
        return oki_riibou2

    #start_East のセッター
    def set_start_East(self, start_East):
        self.start_East = start_East
```

<Yonin.py>

```
# -*- coding: utf-8 -*-
from Taku import Taku
from Jansi import Jansi
from Computer import Computer
from mahjong.constants import EAST, SOUTH, WEST, NORTH
from Kanji import Kanji
from ComAI1 import ComAI1
from ComAI2 import ComAI2
import copy

#四人麻雀のゲーム進行に関するクラス
class Yonin(object):

    def play_game(self):
        taku = Taku()
        comAI1 = ComAI1()
        comAI2 = ComAI2()
        jansi1 = Computer(taku, "コンピュータ 1", comAI2)
        jansi2 = Computer(taku, "コンピュータ 2", comAI1)
        jansi3 = Computer(taku, "コンピュータ 3", comAI1)
        jansi4 = Computer(taku, "コンピュータ 4", comAI1)
        jansi1.set_teki([jansi2,jansi3,jansi4])
        jansi2.set_teki([jansi3,jansi4,jansi1])
        jansi3.set_teki([jansi4,jansi1,jansi2])
        jansi4.set_teki([jansi1,jansi2,jansi3])
        #東からスタートした jansi を入れる
        taku.set_start_East(jansi1)
        jansi1.set_player_wind(EAST)
        jansi2.set_player_wind(SOUTH)
        jansi3.set_player_wind(WEST)
```

```
jansi4.set_player_wind(NORTH)
#現在進行中の雀士
now_jansi = jansi1
#現在進行中の的
now_teki = [jansi2, jansi3, jansi4]
#半荘を終わらせるための関数
hanchan = 0
#一半荘する時の処理
while True:
    #親が変わったら 1 局増やす
    if taku.honba == 0:
        hanchan += 1
    #対局８回かハコテンで終わらせる、全員が 30000 点以下なら続ける
    if jansi1.get_tenbou() < 0 or jansi2.get_tenbou() < 0 or¥
            jansi3.get_tenbou() < 0 or jansi4.get_tenbou() < 0:
        print("場のリーチ棒: ",end='')
        print(taku.oki_riibou)
        self.result_round(jansi1, jansi2, jansi3, jansi4)
        break
    elif hanchan > 8:
        if jansi1.get_tenbou() < 30000 and jansi2.get_tenbou() < 30000 and¥
                jansi3.get_tenbou() < 30000 and jansi4.get_tenbou() < 30000:
            pass
        else:
            print("場のリーチ棒: ",end='')
            print(taku.oki_riibou)
            print(taku.oki_riibou + jansi1.get_tenbou() + jansi2.get_tenbou()¥
                    + jansi3.get_tenbou() + jansi4.get_tenbou())
            self.result_round(jansi1, jansi2, jansi3, jansi4)
            break
    print("--" *30)
    #4 の倍数で場風を変える
    if taku.get_round() % 4 == 0 and taku.get_round() != 0:
        #局数をリセット
        taku.reset_round()
        #場風を変える
        taku.change_round_wind()
    #親がはじめに行動する
    if jansi1.get_player_wind() == EAST:
```

```
        #現在進行中の雀士
        now_jansi = jansi1
        #現在進行中の的
        now_teki = [jansi2, jansi3, jansi4]
elif jansi2.get_player_wind() == EAST:
        #現在進行中の雀士
        now_jansi = jansi2
        #現在進行中の的
        now_teki = [jansi3, jansi4, jansi1]
elif jansi3.get_player_wind() == EAST:
        #現在進行中の雀士
        now_jansi = jansi3
        #現在進行中の的
        now_teki = [jansi4, jansi1, jansi2]
elif jansi4.get_player_wind() == EAST:
        #現在進行中の雀士
        now_jansi = jansi4
        #現在進行中の的
        now_teki = [jansi1, jansi2, jansi3]
#ドラのリセット
taku.reset_dora()
#山のリセット
taku.yama_reset()
#ドラをめくる
taku.add_dora()
#配牌する
jansi1.haipai()
jansi1.riipai()
jansi2.haipai()
jansi2.riipai()
jansi3.haipai()
jansi3.riipai()
jansi4.haipai()
jansi4.riipai()
#鳴きフラグ(誰かが鳴いたら)
naki = False
#ロンしたかどうか
is_ron = False
#１局打つときの処理
```

```
while True:
    #残りの山の数が 14 枚でループを抜ける
    if len(taku.get_yama()) <= 14¥
            or jansi1.get_agari_flag() == True¥
            or jansi2.get_agari_flag() == True¥
            or jansi3.get_agari_flag() == True¥
            or jansi4.get_agari_flag() == True:
        print("")
        break
    else:
        print("--"*30)
    #誰も鳴いてないなら
    if naki == False:
        #局数の表示
        taku.show_round()
        #本場の表示
        taku.show_honba()
        #ドラを表示
        taku.show_dora()
        #山の数
        taku.show_yama()
        print("")
        #現在の雀士の挙動
        now_jansi.game()
        print("--" *30)
        #安全パイに捨て牌を入れる
        if now_jansi == jansi1:
            taku.jansi1_anzenpai.append(now_jansi.get_sutehai()[-1])
        elif now_jansi == jansi2:
            taku.jansi2_anzenpai.append(now_jansi.get_sutehai()[-1])
        elif now_jansi == jansi3:
            taku.jansi3_anzenpai.append(now_jansi.get_sutehai()[-1])
        elif now_jansi == jansi4:
            taku.jansi4_anzenpai.append(now_jansi.get_sutehai()[-1])
        #リーチしているなら打牌を安全パイとして入れる
        if jansi1.get_is_riichi() == True:
            taku.jansi1_riichi == True
            taku.jansi1_anzenpai.append(now_jansi.get_sutehai()[-1])
        elif jansi2.get_is_riichi() == True:
```

```
                taku.jansi2_riichi == True
                taku.jansi2_anzenpai.append(now_jansi.get_sutehai()[-1])
            elif jansi3.get_is_riichi() == True:
                taku.jansi3_riichi == True
                taku.jansi3_anzenpai.append(now_jansi.get_sutehai()[-1])
            elif jansi4.get_is_riichi() == True:
                taku.jansi4_riichi == True
                taku.jansi4_anzenpai.append(now_jansi.get_sutehai()[-1])
            #それぞれの雀士に安全パイを入れる
            jansi1.set_anzenpai(taku.jansi1_anzenpai)
            jansi2.set_anzenpai(taku.jansi2_anzenpai)
            jansi3.set_anzenpai(taku.jansi3_anzenpai)
            jansi4.set_anzenpai(taku.jansi4_anzenpai)
            #卓上に捨て牌を置く
            taku.jansi1_sutehai = jansi1.get_sutehai()
            taku.jansi2_sutehai = jansi2.get_sutehai()
            taku.jansi3_sutehai = jansi3.get_sutehai()
            taku.jansi4_sutehai = jansi4.get_sutehai()
            #卓上に鳴き牌を置く(カカンや暗カンのため)
            self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
        if now_jansi.get_agari_flag() == False:
            sutehai = copy.deepcopy(now_jansi.get_sutehai()[-1])
            #誰かがロンしたかどうか
            is_ron = False
            for i in range(3):
                if is_ron == False:
                    #誰かが捨て牌をロンするかどうか
                    is_ron = self.ron(now_teki[i], now_jansi, taku)
                else:
                    self.ron(now_teki[i], now_jansi, taku)
            #明カンするかどうか
            if is_ron == False:
                naki = now_teki[0].minkan(sutehai)
                if naki == True:
                    #他の人が鳴いたかどうかを入れる
                    now_jansi.naki = True
                    now_teki[0].naki = True
                    now_teki[1].naki = True
                    now_teki[2].naki = True
```

```
                #捨て牌を回収
                del now_jansi.get_sutehai()[-1]
                #鳴いた後の現在の雀士と敵を変える
                now_jansi = now_teki[0]
                now_teki = [now_teki[1], now_teki[2], now_jansi]
                self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
                continue
        if is_ron == False:
            naki = now_teki[1].minkan(sutehai)
            if naki == True:
                #他の人が鳴いたかどうかを入れる
                now_jansi.naki = True
                now_teki[0].naki = True
                now_teki[1].naki = True
                now_teki[2].naki = True
                #捨て牌を回収
                del now_jansi.get_sutehai()[-1]
                #鳴いた後の現在の雀士と敵を変える
                now_jansi = now_teki[1]
                now_teki = [now_teki[2], now_jansi, now_teki[0]]
                self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
                continue
        if is_ron == False:
            naki = now_teki[2].minkan(sutehai)
            if naki == True:
                #他の人が鳴いたかどうかを入れる
                now_jansi.naki = True
                now_teki[0].naki = True
                now_teki[1].naki = True
                now_teki[2].naki = True
                #捨て牌を回収
                del now_jansi.get_sutehai()[-1]
                #鳴いた後の現在の雀士と敵を変える
                now_jansi = now_teki[2]
                now_teki = [now_jansi, now_teki[0], now_teki[1]]
                self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
                continue
        #ポンするかどうか
        if is_ron == False:
```

```
        naki = now_teki[0].pon(sutehai)
        if naki == True:
            #他の人が鳴いたかどうかを入れる
            now_jansi.naki = True
            now_teki[0].naki = True
            now_teki[1].naki = True
            now_teki[2].naki = True
            #捨て牌を回収
            del now_jansi.get_sutehai()[-1]
            #鳴いた後の現在の雀士と敵を変える
            now_jansi = now_teki[0]
            now_teki = [now_teki[1], now_teki[2], now_jansi]
            self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
            continue
    if is_ron == False:
        naki = now_teki[1].pon(sutehai)
        if naki == True:
            #他の人が鳴いたかどうかを入れる
            now_jansi.naki = True
            now_teki[0].naki = True
            now_teki[1].naki = True
            now_teki[2].naki = True
            #捨て牌を回収
            del now_jansi.get_sutehai()[-1]
            #鳴いた後の現在の雀士と敵を変える
            now_jansi = now_teki[1]
            now_teki = [now_teki[2], now_jansi, now_teki[0]]
            self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
            continue
    if is_ron == False:
        naki = now_teki[2].pon(sutehai)
        if naki == True:
            #他の人が鳴いたかどうかを入れる
            now_jansi.naki = True
            now_teki[0].naki = True
            now_teki[1].naki = True
            now_teki[2].naki = True
            #捨て牌を回収
            del now_jansi.get_sutehai()[-1]
```

```
                        #鳴いた後の現在の雀士と敵を変える
                        now_jansi = now_teki[2]
                        now_teki = [now_jansi, now_teki[0], now_teki[1]]
                        self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
                        continue
                #チーするかどうか(上家の人からのみ)
                if is_ron == False:
                    naki = now_teki[0].chii(sutehai)
                    if naki == True:
                        #他の人が鳴いたかどうかを入れる
                        now_jansi.naki = True
                        now_teki[0].naki = True
                        now_teki[1].naki = True
                        now_teki[2].naki = True
                        #捨て牌を回収
                        del now_jansi.get_sutehai()[-1]
                        #now_jansi と now_teki の変更
                        now_jansi = now_teki[0]
                        now_teki = [now_teki[1], now_teki[2], now_jansi]
                        self.nakihai(taku,jansi1,jansi2,jansi3,jansi4)
                        continue
            now_jansi = self.change_jansi(jansi1, jansi2, jansi3, jansi4, now_jansi)
            now_teki = self.change_teki(jansi1, jansi2, jansi3, jansi4, now_jansi)
            naki = False
        #一局終了時の処理
        self.finish(taku, jansi1, jansi2, jansi3, jansi4, is_ron)
junni = 4
#同点だった場合、最初に東だった人から順番に順位を決める
if taku.start_East == jansi1:
    if jansi1.get_tenbou() >= jansi2.get_tenbou():
        junni -= 1
    if jansi1.get_tenbou() >= jansi3.get_tenbou():
        junni -= 1
    if jansi1.get_tenbou() >= jansi4.get_tenbou():
        junni -= 1
elif taku.start_East == jansi2:
    if jansi1.get_tenbou() > jansi2.get_tenbou():
        junni -= 1
    if jansi1.get_tenbou() > jansi3.get_tenbou():
```

```
                junni -= 1
            if jansi1.get_tenbou() > jansi4.get_tenbou():
                junni -= 1
        elif taku.start_East == jansi3:
            if jansi1.get_tenbou() >= jansi2.get_tenbou():
                junni -= 1
            if jansi1.get_tenbou() > jansi3.get_tenbou():
                junni -= 1
            if jansi1.get_tenbou() > jansi4.get_tenbou():
                junni -= 1
        elif taku.start_East == jansi4:
            if jansi1.get_tenbou() >= jansi2.get_tenbou():
                junni -= 1
            if jansi1.get_tenbou() >= jansi3.get_tenbou():
                junni -= 1
            if jansi1.get_tenbou() > jansi4.get_tenbou():
                junni -= 1
        print("プレイヤー１の順位は",end = '')
        print(junni,end='')
        print("でした")
        tenbou = jansi1.get_tenbou()
        junni_tensuu = [junni, tenbou]
        return junni_tensuu

#現在の雀士を変える
def change_jansi(self, jansi1, jansi2, jansi3, jansi4, now_jansi):
        if now_jansi == jansi1:
            return jansi2
        elif now_jansi == jansi2:
            return jansi3
        elif now_jansi == jansi3:
            return jansi4
        elif now_jansi == jansi4:
            return jansi1

#現在の敵の雀士を変える
def change_teki(self, jansi1, jansi2, jansi3, jansi4, now_jansi):
        if now_jansi == jansi1:
            return [jansi2, jansi3, jansi4]
```

```
        elif now_jansi == jansi2:
            return [jansi3, jansi4, jansi1]
        elif now_jansi == jansi3:
            return [jansi4, jansi1, jansi2]
        elif now_jansi == jansi4:
            return [jansi1, jansi2, jansi3]

#鳴き牌を卓上に置く
def nakihai(self, taku, jansi1, jansi2, jansi3, jansi4):
        taku.jansi1_nakihai = jansi1.unit_nakihai()
        taku.jansi2_nakihai = jansi2.unit_nakihai()
        taku.jansi3_nakihai = jansi3.unit_nakihai()
        taku.jansi4_nakihai = jansi4.unit_nakihai()

#ロンのときの処理
def ron(self, now_teki, now_jansi, taku):
        now_teki.ron(now_jansi.get_sutehai()[-1], now_jansi.name)
        if now_teki.agari_flag == True:
            now_jansi.sub_tenbou(now_teki.get_move_tenbou()[0] + taku.get_honba() * 300)
            #上がった場合捨て牌をもらう
            del now_teki.get_sutehai()[-1]
            return True
        return False


# 1 局終了時の処理
def finish(self, taku, jansi1, jansi2, jansi3, jansi4, is_ron):
            dareka_agari = False
            #誰かロンしたなら
            if is_ron == True:
                dareka_agari = True
            #誰か上がったら dareka_agari を True にする
            if (jansi1.get_agari_flag() == True or jansi2.get_agari_flag() == True or¥
                    jansi3.get_agari_flag() == True or jansi4.get_agari_flag() == True) and is_ron == False:
                dareka_agari = True
                #雀士 1 が上がった時
                if jansi1.get_agari_flag() == True:
                    if jansi1.get_player_wind() == EAST:
                        jansi2.sub_tenbou(jansi1.get_move_tenbou()[0] + taku.get_honba()*100)
```

```
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[0] + taku.get_honba()*100)
        #雀士２が親なら雀士２が親被り、他は子の点数引かれる
        elif jansi2.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[1] + taku.get_honba()*100)
        #雀士 3 が親被り
        elif jansi3.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[1] + taku.get_honba()*100)
        #雀士 4 が親被り
        elif jansi4.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[0] + taku.get_honba()*100)
    #雀士 2 が上がった時
    elif jansi2.get_agari_flag() == True:
        if jansi2.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[0] + taku.get_honba()*100)
        #雀士 3 が親なら雀士 3 が親被り、他は子の点数引かれる
        elif jansi3.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[1] + taku.get_honba()*100)
        #雀士 4 が親被り
        elif jansi4.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[0] + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[1] + taku.get_honba()*100)
        #雀士 1 が親被り
        elif jansi1.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[1] + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[0] + taku.get_honba()*100)
    #雀士 3 が上がった時
```

```
elif jansi3.get_agari_flag() == True:
    if jansi3.get_player_wind() == EAST:
        jansi4.sub_tenbou(jansi3.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi1.sub_tenbou(jansi3.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi3.get_move_tenbou()[0] + taku.get_honba()*100)
    #雀士 4 が親なら雀士 4 が親被り、他は子の点数引かれる
    elif jansi4.get_player_wind() == EAST:
        jansi4.sub_tenbou(jansi3.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi1.sub_tenbou(jansi3.get_move_tenbou()[1] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi3.get_move_tenbou()[1] + taku.get_honba()*100)
    #雀士 1 が親被り
    elif jansi1.get_player_wind() == EAST:
        jansi4.sub_tenbou(jansi3.get_move_tenbou()[1] + taku.get_honba()*100)
        jansi1.sub_tenbou(jansi3.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi3.get_move_tenbou()[1] + taku.get_honba()*100)
    #雀士 2 が親被り
    elif jansi2.get_player_wind() == EAST:
        jansi4.sub_tenbou(jansi3.get_move_tenbou()[1] + taku.get_honba()*100)
        jansi1.sub_tenbou(jansi3.get_move_tenbou()[1] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi3.get_move_tenbou()[0] + taku.get_honba()*100)
#雀士 4 が上がった時
elif jansi4.get_agari_flag() == True:
    if jansi4.get_player_wind() == EAST:
        jansi1.sub_tenbou(jansi4.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi4.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi3.sub_tenbou(jansi4.get_move_tenbou()[0] + taku.get_honba()*100)
    #雀士 1 が親なら雀士 1 が親被り、他は子の点数引かれる
    elif jansi1.get_player_wind() == EAST:
        jansi1.sub_tenbou(jansi4.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi4.get_move_tenbou()[1] + taku.get_honba()*100)
        jansi3.sub_tenbou(jansi4.get_move_tenbou()[1] + taku.get_honba()*100)
    #雀士 2 が親被り
    elif jansi2.get_player_wind() == EAST:
        jansi1.sub_tenbou(jansi4.get_move_tenbou()[1] + taku.get_honba()*100)
        jansi2.sub_tenbou(jansi4.get_move_tenbou()[0] + taku.get_honba()*100)
        jansi3.sub_tenbou(jansi4.get_move_tenbou()[1] + taku.get_honba()*100)
    #雀士 3 が親被り
    elif jansi3.get_player_wind() == EAST:
        jansi1.sub_tenbou(jansi4.get_move_tenbou()[1] + taku.get_honba()*100)
```

```
                jansi2.sub_tenbou(jansi4.get_move_tenbou()[1] + taku.get_honba()*100)
                jansi3.sub_tenbou(jansi4.get_move_tenbou()[0] + taku.get_honba()*100)
# 1 局終了時の処理
nagasi1 = jansi1.finish_round(dareka_agari)
nagasi2 = jansi2.finish_round(dareka_agari)
nagasi3 = jansi3.finish_round(dareka_agari)
nagasi4 = jansi4.finish_round(dareka_agari)
#流し満貫用
if nagasi1 == True or nagasi2 == True or nagasi3 == True or nagasi4 == True:
    if nagasi1 == True:
        if jansi1.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[0]/3 + taku.get_honba()*100)
        elif jansi2.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi3.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi4.get_player_wind() == EAST:
            jansi2.sub_tenbou(jansi1.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi1.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi1.get_move_tenbou()[0]/2 + taku.get_honba()*100)
    if nagasi2 == True:
        if jansi2.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[0]/3 + taku.get_honba()*100)
        elif jansi3.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi4.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[0]/4 + taku.get_honba()*100)
```

```
        elif jansi1.get_player_wind() == EAST:
            jansi3.sub_tenbou(jansi2.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi4.sub_tenbou(jansi2.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi2.get_move_tenbou()[0]/2 + taku.get_honba()*100)
    if nagasi3 == True:
        if jansi3.get_player_wind() == EAST:
            jansi4.sub_tenbou(jansi3.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi3.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi3.get_move_tenbou()[0]/3 + taku.get_honba()*100)
        elif jansi4.get_player_wind() == EAST:
            jansi4.sub_tenbou(jansi3.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi3.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi3.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi1.get_player_wind() == EAST:
            jansi4.sub_tenbou(jansi3.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi3.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi3.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi2.get_player_wind() == EAST:
            jansi4.sub_tenbou(jansi3.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi1.sub_tenbou(jansi3.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi3.get_move_tenbou()[0]/2 + taku.get_honba()*100)
    if nagasi4 == True:
        if jansi4.get_player_wind() == EAST:
            jansi1.sub_tenbou(jansi4.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi4.get_move_tenbou()[0]/3 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi4.get_move_tenbou()[0]/3 + taku.get_honba()*100)
        elif jansi1.get_player_wind() == EAST:
            jansi1.sub_tenbou(jansi4.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi4.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi4.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi2.get_player_wind() == EAST:
            jansi1.sub_tenbou(jansi4.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi4.get_move_tenbou()[0]/2 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi4.get_move_tenbou()[0]/4 + taku.get_honba()*100)
        elif jansi3.get_player_wind() == EAST:
            jansi1.sub_tenbou(jansi4.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi2.sub_tenbou(jansi4.get_move_tenbou()[0]/4 + taku.get_honba()*100)
            jansi3.sub_tenbou(jansi4.get_move_tenbou()[0]/2 + taku.get_honba()*100)
#誰も上がってない時聴牌時の処理
```

```
if jansi1.get_agari_flag() == False and jansi2.get_agari_flag() == False and¥
        jansi3.get_agari_flag() == False and jansi4.get_agari_flag() == False:
    if jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == False and¥
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == False:
        jansi1.add_tenbou(3000)
        jansi2.sub_tenbou(1000)
        jansi3.sub_tenbou(1000)
        jansi4.sub_tenbou(1000)
    elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == True and¥
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == False:
        jansi1.sub_tenbou(1000)
        jansi2.add_tenbou(3000)
        jansi3.sub_tenbou(1000)
        jansi4.sub_tenbou(1000)
    elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == False and¥
            jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == False:
        jansi1.sub_tenbou(1000)
        jansi2.sub_tenbou(1000)
        jansi3.add_tenbou(3000)
        jansi4.sub_tenbou(1000)
    elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == False and¥
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == True:
        jansi1.sub_tenbou(1000)
        jansi2.sub_tenbou(1000)
        jansi3.sub_tenbou(1000)
        jansi4.add_tenbou(3000)
    elif jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == True and¥
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == False:
        jansi1.add_tenbou(1500)
        jansi2.add_tenbou(1500)
        jansi3.sub_tenbou(1500)
        jansi4.sub_tenbou(1500)
    elif jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == False and¥
            jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == False:
        jansi1.add_tenbou(1500)
        jansi2.sub_tenbou(1500)
        jansi3.add_tenbou(1500)
        jansi4.sub_tenbou(1500)
    elif jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == False and¥
```

```
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == True:
        jansi1.add_tenbou(1500)
        jansi2.sub_tenbou(1500)
        jansi3.sub_tenbou(1500)
        jansi4.add_tenbou(1500)
    elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == True and¥
            jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == False:
        jansi1.sub_tenbou(1500)
        jansi2.add_tenbou(1500)
        jansi3.add_tenbou(1500)
        jansi4.sub_tenbou(1500)
    elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == True and¥
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == True:
        jansi1.sub_tenbou(1500)
        jansi2.add_tenbou(1500)
        jansi3.sub_tenbou(1500)
        jansi4.add_tenbou(1500)
    elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == False and¥
            jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == True:
        jansi1.sub_tenbou(1500)
        jansi2.sub_tenbou(1500)
        jansi3.add_tenbou(1500)
        jansi4.add_tenbou(1500)
    elif jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == True and¥
            jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == False:
        jansi1.add_tenbou(1000)
        jansi2.add_tenbou(1000)
        jansi3.add_tenbou(1000)
        jansi4.sub_tenbou(3000)
    elif jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == True and¥
            jansi3.get_tenpai_flag() == False and jansi4.get_tenpai_flag() == True:
        jansi1.add_tenbou(1000)
        jansi2.add_tenbou(1000)
        jansi3.sub_tenbou(3000)
        jansi4.add_tenbou(1000)
    elif jansi1.get_tenpai_flag() == True and jansi2.get_tenpai_flag() == False and¥
            jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == True:
        jansi1.add_tenbou(1000)
        jansi2.sub_tenbou(3000)
```

```
            jansi3.add_tenbou(1000)
            jansi4.add_tenbou(1000)
        elif jansi1.get_tenpai_flag() == False and jansi2.get_tenpai_flag() == True and¥
                jansi3.get_tenpai_flag() == True and jansi4.get_tenpai_flag() == True:
            jansi1.sub_tenbou(3000)
            jansi2.add_tenbou(1000)
            jansi3.add_tenbou(1000)
            jansi4.add_tenbou(1000)
#親が和了かテンパイなら本場を増やす、皆ノーテンなら流す
if (jansi1.get_player_wind() == EAST and jansi1.get_agari_flag() == True) or¥
    (jansi1.get_player_wind() == EAST and jansi1.get_tenpai_flag() == True) or¥
    (jansi2.get_player_wind() == EAST and jansi2.get_agari_flag() == True) or¥
    (jansi2.get_player_wind() == EAST and jansi2.get_tenpai_flag() == True) or¥
    (jansi3.get_player_wind() == EAST and jansi3.get_agari_flag() == True) or¥
    (jansi3.get_player_wind() == EAST and jansi3.get_tenpai_flag() == True) or¥
    (jansi4.get_player_wind() == EAST and jansi4.get_agari_flag() == True) or¥
    (jansi4.get_player_wind() == EAST and jansi4.get_tenpai_flag() == True):
    #本場を増やす
    taku.add_honba()
else:
    #自風を変える
    jansi1.change_player_wind()
    jansi2.change_player_wind()
    jansi3.change_player_wind()
    jansi4.change_player_wind()
    #局数を増やす
    taku.add_round()
    #本場を 0 にする
    taku.reset_honba()
#上がりのフラグをリセットする
jansi1.reset_agari_flag()
jansi2.reset_agari_flag()
jansi3.reset_agari_flag()
jansi4.reset_agari_flag()
#テンパイのフラグをリセットする
jansi1.tenpai_flag = False
jansi2.tenpai_flag = False
jansi3.tenpai_flag = False
jansi4.tenpai_flag = False
```

```
            print("")
            jansi1.show_name()
            jansi1.show_tenbou()
            print("")
            jansi2.show_name()
            jansi2.show_tenbou()
            print("")
            jansi3.show_name()
            jansi3.show_tenbou()
            print("")
            jansi4.show_name()
            jansi4.show_tenbou()
            print("")
            print("##" * 35)
            #公開されてる捨て牌と鳴き牌を初期化
            taku.jansi1_nakihai = []
            taku.jansi2_nakihai = []
            taku.jansi3_nakihai = []
            taku.jansi4_nakihai = []
            taku.jansi1_sutehai = []
            taku.jansi2_sutehai = []
            taku.jansi3_sutehai = []
            taku.jansi4_sutehai = []
            taku.jansi1_riichi = False
            taku.jansi2_riichi = False
            taku.jansi3_riichi = False
            taku.jansi4_riichi = False
            taku.jansi1_anzenpai = []
            taku.jansi2_anzenpai = []
            taku.jansi3_anzenpai = []
            taku.jansi4_anzenpai = []

#対局結果の表示
def result_round(self, jansi1, jansi2, jansi3, jansi4):
    print("")
    #持ち点を表示
    print("対局結果")
    print("")
    jansi1.show_name()
```

```
            jansi1.show_tenbou()
            print("")
            jansi2.show_name()
            jansi2.show_tenbou()
            print("")
            jansi3.show_name()
            jansi3.show_tenbou()
            print("")
            jansi4.show_name()
            jansi4.show_tenbou()
            print("")
            print("")
            print("##"*35)


if __name__ == "__main__":
    junni = []
    tensuu = []
    while True:
            if len(junni) < 300:
                try:
                    print(len(junni)+1,end='')
                    print("回目")
                    junni_tensuu = Yonin().play_game()
                    junni.append(junni_tensuu[0])
                    tensuu.append(junni_tensuu[1])
                except:
                    pass
            else:
                break
    value1 = 0
    value2 = 0
    value3 = 0
    value4 = 0
    for i in junni:
        if i == 1:
            value1 += 1
        elif i == 2:
            value2 += 1
```

```
        elif i == 3:
            value3 += 1
        elif i == 4:
            value4 += 1
print("")
print("--"*15,end="")
print("コンピュータ 1 の結果",end="")
print("--"*15)
print("条件")
"""
print("・他家がリーチする → オリる")
"""
print("・16 枚牌を捨てる → オリる")
"""
print("自分が 2 向聴の場合のみ → オリない")
"""
print("・自分が 4 位で一向聴の場合のみ → オリない")

print("・4 位以下がリーチした場合のみ → オリる")
"""
print("・ただし 4 位との点差が 2000 点 → オリない")
""
print("・親なら　→　降りない")
"""
print("合計回数: ",end="")
print(len(junni))
print("1 位：  ",end="")
print(value1)
print("2 位：  ",end="")
print(value2)
print("3 位：  ",end="")
print(value3)
print("4 位：  ",end="")
print(value4)
print("最下位率: ",end="")
try:
    print(value4/(value1+value2+value3+value4),end="")
    print("%")
except:
```

```
        print("0%")
print("平均点数: ",end='')
print(sum(tensuu)/len(tensuu),end='')
print("点")
```