### Dokumentacja Klasy BluffStateEngine

#### Opis:

Klasa BluffStateEngine reprezentuje silnik stanu gry dla gry w bluff. Zarządza stanem planszy i stanem gry oraz obsługuje zmiany stanów gry.

#### Metody Publiczne:

1. constructor(options: Readonly<IGameOptions>, stateTransitions: IStateTransition[], stateTransitionCallback: FStateTransitionCallback)
   * Opis: Konstruktor klasy BluffStateEngine, inicjalizuje stan planszy i stan gry oraz definiuje przejścia stanów.
   * Parametry:
     + options: Readonly<IGameOptions> - Opcje gry.
     + stateTransitions: IStateTransition[] - Tablica obiektów definiujących możliwe przejścia stanów.
     + stateTransitionCallback: FStateTransitionCallback - Funkcja wywoływana po wykonaniu przejścia.
2. changeState(nextState: IBluffGameStates, move?: IBluffMove)
   * Opis: Metoda zmienia stan gry na podany stan.
   * Parametry:
     + nextState: IBluffGameStates - Następny stan gry.
     + move?: IBluffMove - Ruch wykonany przez gracza (opcjonalny).
   * Wyjątek:
     + Zgłasza błąd w przypadku braku zdefiniowanego przejścia między stanami.
3. getBoardState(): IBoardState
   * Opis: Metoda zwraca stan planszy.
   * Zwraca: Stan planszy.
4. getGameState(): IGameState
   * Opis: Metoda zwraca stan gry.
   * Zwraca: Stan gry.

#### Właściwości Prywatne:

1. boardState: IBoardState - Przechowuje stan planszy.
2. gameState: IGameState - Przechowuje stan gry.

#### Importy:

* FStateTransition, FStateTransitionCallback, IStateTransition, IStateTransitionProps z "./BluffStateEngineTransitionTypes"
* gameEngineHelper z "./GameEngineHelper"
* IBluffGameStates, IBluffMove, IBoardState, IGameOptions, IGameState z "./IGameStateTypes"

### Użycie Klasy BluffStateEngine:

import { BluffStateEngine } from "./BluffStateEngine";  
  
// Inicjalizacja silnika stanu gry  
const stateEngine = new BluffStateEngine(options, stateTransitions, stateTransitionCallback);  
  
// Zmiana stanu gry  
stateEngine.changeState('NextState');  
  
// Pobranie stanu planszy  
const currentBoardState = stateEngine.getBoardState();  
  
// Pobranie stanu gry  
const currentGameState = stateEngine.getGameState();

Klasa BluffStateEngine zapewnia obsługę stanów gry i zarządzanie przejściami między nimi w grze w bluf. Dzięki niej możliwe jest dynamiczne zmienianie stanów gry zgodnie z logiką definowaną w obiektach stateTransitions, reagowanie na ruchy graczy i synchronizowanie stanu planszy oraz stanu gry.

### Dokumentacja dla Obiektu bluffStateEngineTransitions

#### Opis:

Obiekt bluffStateEngineTransitions zawiera tablicę definicji przejść stanów dla silnika stanu gry w grze w bluff. Każda definicja określa aktualny stan, następny stan oraz funkcję przejścia między nimi.

#### Struktura:

* currentState: string - Aktualny stan gry.
* nextState: string - Następny stan gry.
* makeTransition?: (props: IStateTransitionProps) => void - Opcjonalna funkcja wykonująca przejście ze stanu currentState do nextState.

#### Definicje Przejść:

1. NEW\_GAME => NEW\_ROUND
   * Opis: Przejście do nowej rundy.
   * Funkcja Przejścia: newRoundTransition
2. NEW\_ROUND => WAITING\_FOR\_PLAYER\_MOVE
   * Opis: Oczekiwanie na ruch gracza.
   * Funkcja Przejścia: waitingForPlayerTransition
3. WAITING\_FOR\_PLAYER\_MOVE => PLAYER\_MOVE\_BID
   * Opis: Gracz wykonuje licytację.
   * Funkcja Przejścia: playerMoveBidTransition
4. PLAYER\_MOVE\_BID => WAITING\_FOR\_PLAYER\_MOVE
   * Opis: Powrót do oczekiwania na ruch gracza.
   * Funkcja Przejścia: waitingForPlayerTransition
5. WAITING\_FOR\_PLAYER\_MOVE => PLAYER\_MOVE\_CHALLENGE
   * Opis: Gracz stawia wyzwanie.
   * Funkcja Przejścia: playerMoveChallangeTransition
6. PLAYER\_MOVE\_CHALLENGE => END\_ROUND
   * Opis: Zakończenie rundy po wyzwaniu gracza.
7. END\_ROUND => END\_GAME
   * Opis: Zakończenie gry po zakończeniu rundy.
8. END\_ROUND => NEW\_ROUND
   * Opis: Powrót do nowej rundy po zakończeniu rundy.
   * Funkcja Przejścia: newRoundTransition

#### Funkcje Przejść:

1. newRoundTransition(props: IStateTransitionProps)
   * Opis: Funkcja inicjalizująca nową rundę ustawiająca historię licytacji i wartości kostek.
   * Parametry:
     + props: IStateTransitionProps - Obiekt zawierający stan planszy, stan gry i opcje gry.
2. playerMoveBidTransition(props: IStateTransitionProps)
   * Opis: Funkcja obsługująca ruch gracza typu licytacja, dodaje licytację do historii.
   * Parametry:
     + props: IStateTransitionProps - Obiekt zawierający stan planszy, stan gry, ruch gracza i opcje gry.
3. playerMoveChallangeTransition(props: IStateTransitionProps)
   * Opis: Funkcja obsługująca ruch gracza typu wyzwanie, aktualizuje liczbę kości graczy.
   * Parametry:
     + props: IStateTransitionProps - Obiekt zawierający stan planszy, stan gry i ruch gracza.
4. waitingForPlayerTransition(props: IStateTransitionProps)
   * Opis: Funkcja określająca aktywnego gracza do ruchu na podstawie poprzedniego stanu.
   * Parametry:
     + props: IStateTransitionProps - Obiekt zawierający poprzedni stan, stan gry i opcje gry.

#### Importy:

* IStateTransition, IStateTransitionProps z "./BluffStateEngineTransitionTypes"
* gameEngineHelper z "./GameEngineHelper"

### Zastosowanie Definicji Przejść w Silniku Stanu Gry:

import { bluffStateEngineTransitions } from "./bluffStateEngineTransitions";  
  
// Wykorzystanie definicji przejść w silniku stanu gry do obsługi zmian stanów  
const transition = bluffStateEngineTransitions.find(item => item.currentState === currentState && item.nextState === nextState);  
  
if (transition == null) {  
 throw new Error(`Przejście "${currentState}" => "${nextState}" nie jest zdefiniowane.`)  
}  
  
transition.makeTransition && transition.makeTransition(props);

Obiekt bluffStateEngineTransitions zapewnia strukturyzowane definicje przejść stanów gry oraz funkcje przejścia, które mogą być efektywnie wykorzystane do zarządzania przejściami stanów w silniku stanu gry w grze w bluff. Każde przejście jest jasno określone, co ułatwia zarządzanie logiką gry i zachowaniami graczy.

### Typy i Interfejsy do Zarządzania Przejściami Stanów Gry

W powyższym kodzie znajdują się interfejsy i typy związane z zarządzaniem przejściami stanów gry w kontekście gry w Bluff.

### Interfejsy:

1. IStateTransition:
   * **Pola:**
     + currentState: IBluffGameStates - Aktualny stan gry.
     + nextState: IBluffGameStates - Następny stan gry.
     + makeTransition?: FStateTransition - Opcjonalna funkcja wykonująca przejście między stanami.
2. IStateTransitionCallbackProps:
   * **Pola:**
     + prevState: IBluffGameStates - Stan gry poprzedni.
     + currentState: IBluffGameStates - Aktualny stan gry.
     + boardState: IBoardState - Stan planszy gry.
     + gameState: IGameState - Stan gry.
     + move?: IBluffMove - Ruch gracza (opcjonalny).
3. IStateTransitionProps:
   * **Pola:**
     + options: Readonly<IGameOptions> - Stałe opcje gry.
     + boardState: IBoardState - Stan planszy gry.
     + gameState: IGameState - Stan gry.
     + move?: IBluffMove - Ruch gracza (opcjonalny).
     + prevState: IBluffGameStates - Poprzedni stan gry.
     + currentState: IBluffGameStates - Aktualny stan gry.

### Typy Funkcji:

1. FStateTransitionCallback:
   * **Definicja:**

(props: IStateTransitionCallbackProps) => void

1. FStateTransition:
   * **Definicja:**

(props: IStateTransitionProps) => void

Dzięki powyższym interfejsom i typom, możemy sprawnie zarządzać przejściami stanów gry w kontekście gry w Bluff. Każdy interfejs definiuje niezbędne pola do obsługi stanu gry, ruchów graczy oraz funkcji przejścia między różnymi stanami, co ułatwia implementację logiki gry oraz interakcji pomiędzy różnymi częściami gry. Korzystając z tych abstrakcji, projekt gry może być bardziej modularny i łatwiejszy w utrzymaniu.

#### Funkcje pomocnicze:

1. createBlankBoardState(options: IGameOptions): IBoardState:
   * Tworzy pusty stan planszy z wartościami standardowymi [] dla rzutów kostką i [] dla historii licytacji.
2. initBoardState(options: IGameOptions, gameState: IGameState): IBoardState:
   * Inicjalizuje stan planszy gry przed rozpoczęciem rundy, generując wartości rzutów kostką dla każdego z graczy.
3. createGameState(options: IGameOptions): IGameState:
   * Tworzy nowy stan gry, ustawiając aktywnego gracza na początkowego, przydziela gracza i ustawia stan gry na "NEW\_GAME".
4. nextPlayerIdx(options: IGameOptions, gameState: IGameState): number:
   * Zwraca indeks następnego gracza w kolejce gry na podstawie aktywnego gracza oraz ilości kości graczy.
5. firstPlayerIdx(options: IGameOptions, gameState: IGameState): number:
   * Zwraca indeks pierwszego żywego gracza w grze na podstawie liczby posiadanych kości.
6. isGameFinished(gameState: IGameState): boolean:
   * Sprawdza, czy gra została zakończona na podstawie liczby aktywnych graczy (z ilością większą niż 0).

### Typy i Interfejsy Silnika Gry Bluff

#### Zdefiniowane Typy i Interfejsy:

1. IBoardState:
   * diceValues: number[][] - Wyniki rzutów kości graczy, gdzie gracz wyeliminowany ma pustą tablicę.
   * bidHistory: IPlayerBid[] - Licytacje w obecnej rundzie.
2. IPlayerBid:
   * playerIdx: number - Identyfikator gracza.
   * pickedDice: number - Wybrana kostka.
   * guessedCount: number - Zgadywana ilość kostek.
3. IGameOptions:
   * diceType: number - Typ kostki (domyślnie k6).
   * players: IPlayerAdapter[] - Tablica zawierająca graczy biorących udział w grze.
4. IGameState:
   * activePlayerIdx: number - Indeks aktywnego gracza, liczony od 0.
   * playerDicesCount: number[] - Tablica przechowująca ilość kości posiadanych przez każdego gracza.
   * state: IBluffGameStates - Stan, w którym znajduje się gra.
5. IBluffGameStates:
   * Stan gry określający możliwe etapy gry: "NEW\_GAME", "END\_GAME", "END\_ROUND", "NEW\_ROUND", "WAITING\_FOR\_PLAYER\_MOVE", "PLAYER\_MOVE\_BID", "PLAYER\_MOVE\_CHALLENGE".
6. IBluffMove:
   * Definicja ruchów rozpoznawanych przez BluffGameEngine, takich jak "bid" (z licytacją zawierającą dane IPlayerBid) lub "challenge".