Ludo UML Class Diagram

Ludo

MainWindow

+ MainWindow(): void
+ StartGameHandler(): void, delegate
+ AIHandler(List<LudoColor>): void, delegate
+ ThrowDieHandler(): void, delegate
+ EndTurnHandler(): void, delegate
+ ColoredRoutesHandler(object, object, object, object): void, delegate
+ ColoredBasesHandler(object, object, object, object): void, delegate
+ UIButtonsAndLabelsHandler(object, object, object, object): void, delegate
+ StartGameEvent: event
+ AIEnableEvent: event
+ ThrowDieEvent: event
+ EndTurnEvent: event
+ PrepareColoredRoutes: event
+ PrepareColoredBases: event
+ PrepareBtnsAndLabels: event
+ MovePiece: event
- MovePieceFromAndTo(object, RoutedEventArgs): void
- ThrowDie(object, RoutedEventArgs): void
- EndTurn(object, RoutedEventArgs): void
- StartGame(object, RoutedEventArgs): void
- CheckEnabledAI(object, RoutedEventArgs): void
- ExitApplication(object, RoutedEventArgs): void
- RestartApplication(object, RoutedEventArgs): void
- CreateRoutesAndBases(): void
- SendButtonsAndLabelsToUIManager(): void

LudoClassLibrary

<<enumeration>> LudoColor

Blue, Green, Red, Yellow

<<static> Die

- random: Random
- randomDieValue: int
- hasBeenThrown: bool
- timesThrown: int

~ <<get/set>>TimesThrown(int)
~ <<get/set>>HasBeenThrown(bool)
~ <<get>>RandomDieValue(): int

Field

~ fieldId: string
- isGoal: bool
~ occupants: List<Piece>

~ Field(string): int
~ Field(string, bool): void
~ RemoveOccupant(): void
~ GetOccupantColor(): LudoColor
~<<get>>IsGoal(): bool
~ CountOccupants(): int
~ AddOccupant(Piece): void

Board

~ normalFields: Field[]
~ blueRoute: Field[]
~ greenRoute: Field[]
~ redRoute: Field[]
~ yellowRoute: Field[]
~ blueBase: Field[]
~ greenBase: Field[]
~ redBase: Field[]
~ yellowBase: Field[]

~ InitWholeBoardAndCreateRoutes(): void

Piece

- color: LudoColor
~ routePosition: int
- reachedGoal:bool
- inBase: bool
- basePosition: int

~ Piece(LudoColor, int): void
~<<get>>BasePosition(): int
~<<get>>Color(): LudoColor
~<<get/set>>ReachedGoal(bool)
~<<get/set>>InBase(bool)

GameManager

- currentPlayer: Player
- activePlayersIndex: int
- dieValue: int
- enabledAIPlayers: List<LudoColor>
- activePlayers: List<Player>
- bluePlayer: Player
- greenPlayer: Player
- redPlayer: Player
- yellowPlayer: Player
- bluePieces: List<Piece>
- greenPieces: List<Piece>
- redPieces: List<Piece>
- yellowPieces: List<Piece>
- playerPiecePairs: Dictionary<Player, List<Piece>>
- ludoBoard: Board
+ ui: UIManager

+ RunGame(): void
+ StartNextTurn(): void
- RunAI(): void
+ ThrowDie(): void
- GoToNextPlayer(): void
+ RegisterEnabledAIPlayers(List<LudoColor>): void
- AddPlayersToGame(): void
- ChoosePieceToMove(object fieldPiece, EventArgs): void
- MovePieceFromBaseToRoute(int, Button): void
- MovePieceFromFieldToField(int, Button): void
- WillIPassAllyWithThisMove(LudoColor, int): bool
- FieldHasAPieceOnIt(LudoColor, int): bool
- CanTakeField(int, Piece): bool
- ValidateIfChosenBtnFieldIsPartOfRoute(Button): bool
- MessageNotAValidMove(): void
- MessageYouGotHitBack(): void
- CheckForWinner(): void

<<abstract>> Player

- color: LudoColor
~ hasFinished: bool

~<<get>>Color(): LudoColor

AIPlayer

~ RollDie(bool): int
~ CheckIfAPieceIsInBase(List<Piece>): Piece
~ CheckIfAllPiecesAreInBase(List<Piece>): bool
~ SelectRandomPiece(int, List<Piece>): int
- CheckIfPieceWillPassAlly(int, int, List<Piece>): bool
- PickPieceFurthestAhead(List<Piece>): int

HumanPlayer

UIManager

~ btnDie: Button
~ btnEndTurn: Button
- lblPlayerTurn: Label
- lblInformation: Label
~ btnColorRoutes: Dictionary<LudoColor, List<Button>>
~ btnColorBases: Dictionary<LudoColor, List<Button>>

+ DefineBtnRoutes(object, object, object, object): void
+ DefineBtnBases(object, object, object, object): void
+ InitializeUILabelsAndActionBtns(object, object, object, object): void
~ ShowCurrentPlayer(string): void
~ UpdateInformationLabel(string): void
~ ChangeColor(LudoColor, Button): void
~ ChangeNumberOfOccupants(int, Button): void
- ChangeColorToPlayerColor(LudoColor, Button): void
~ ChangeColorToOriginalColor(Button): void
~ GetIndexOfField(LudoColor, Button): int