Software Requirements Specification

Author: *Levir Heladio Hernandez Suarez*

[Go to Linked**in** *]*   *[Go to GitHub]*

Functional Requirements

The system must:

1. Allow selection of the number of tiles for domino variations.
2. Allow selection of the number of players.
3. Automatically deal tiles to each player at the start of the game.
4. Start the game with the player holding the highest double.
5. Redeal the tiles if no player has a double.
6. Display the tiles on the board for all players.
7. Show the number of tiles in the boneyard for all players.
8. Reveal a player's tiles during their turn, while they can only see how many tiles the other players have.
9. Display the available tiles to play on the board for the player.
10. Allow players to choose which tile to play and which side of the board to place it on.
11. Permit players to draw unknown tiles from the boneyard until they can play them or the boneyard runs out of tiles.
12. Allow players to pass their turn only when the boneyard is empty; otherwise, they must draw from the boneyard.
13. End the game when a player runs out of tiles, declaring them the winner.
14. Determine the winner based on points (the lowest total wins) if all players pass their turns consecutively.
15. Display a summary of the match.

Non-Functional Requirements

The system should:

1. Respond to player actions in under one second.
2. Have an intuitive interface for all player actions.
3. Support future addition of CPU players without requiring changes to existing code.

Use-Case Specifications

UC-1: Starting the Game

- **Title:** Starting the Game
- **Primary Actor:** Main Player [Player]

Success Scenario:

1. The main player selects the number of players and the domino variation.
2. The system automatically deals tiles to all players and places the remaining tiles in the boneyard.
3. The system determines the starting player by identifying the player with the highest double.
4. If no player has a double, the system redeals the tiles.
5. The process repeats until a starting player is determined.

UC-2: Playing the Game

- **Title:** Playing the Game
- **Primary Actor:** Players [Player]

Success Scenario:

1. Each player takes turns placing a tile from their hand on the board, matching one of the open ends.
2. If a player cannot play, they must draw a tile from the boneyard.
3. If the boneyard is empty, the player must pass their turn.

Preconditions:

- The game must have been started, and each player must have a hand of tiles.

UC-3: Ending the Game

Title: Ending the Game

Primary Actor: Winning player [Player]

Success Scenario:

1. A player uses all their tiles and is declared the winner.
2. The system prompts the players to either exit the game or start a new match.

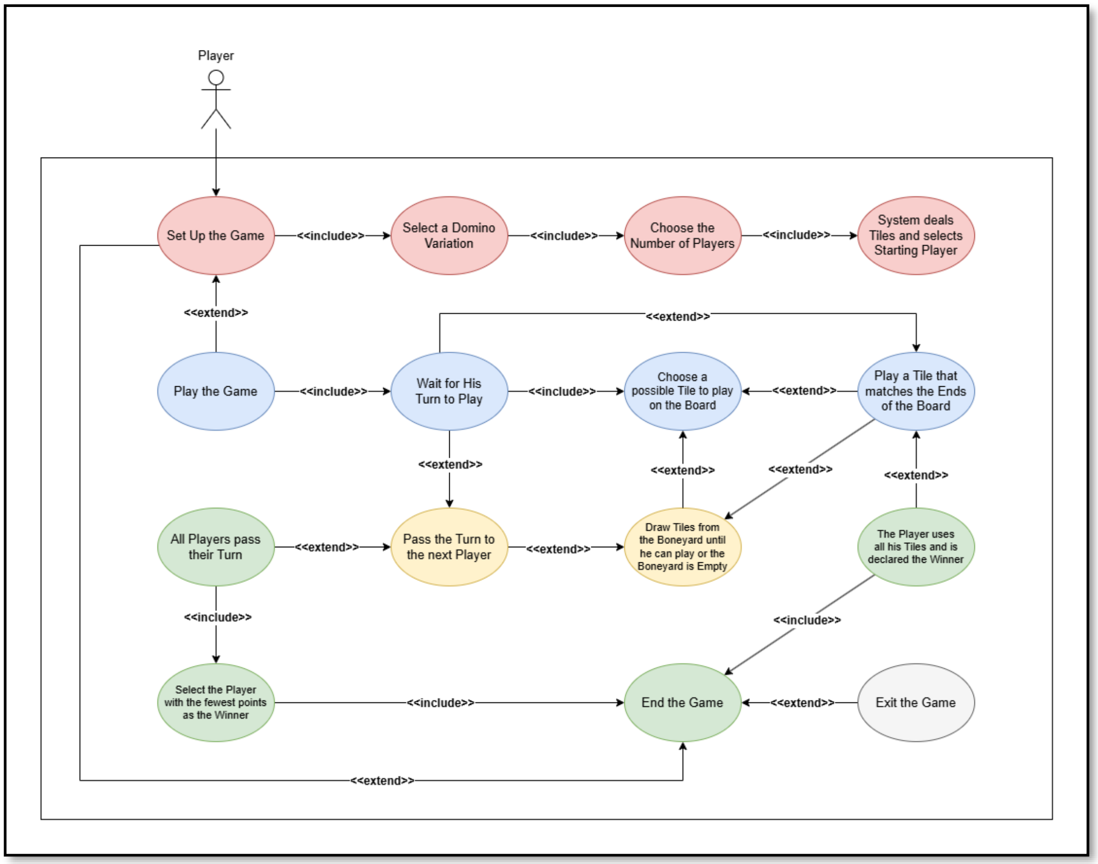
Extensions:

- If the game ends in a draw, the system calculates the winner based on the player with the lowest total points in hand.
- If players choose to start a new match, the system returns to the game setup screen.

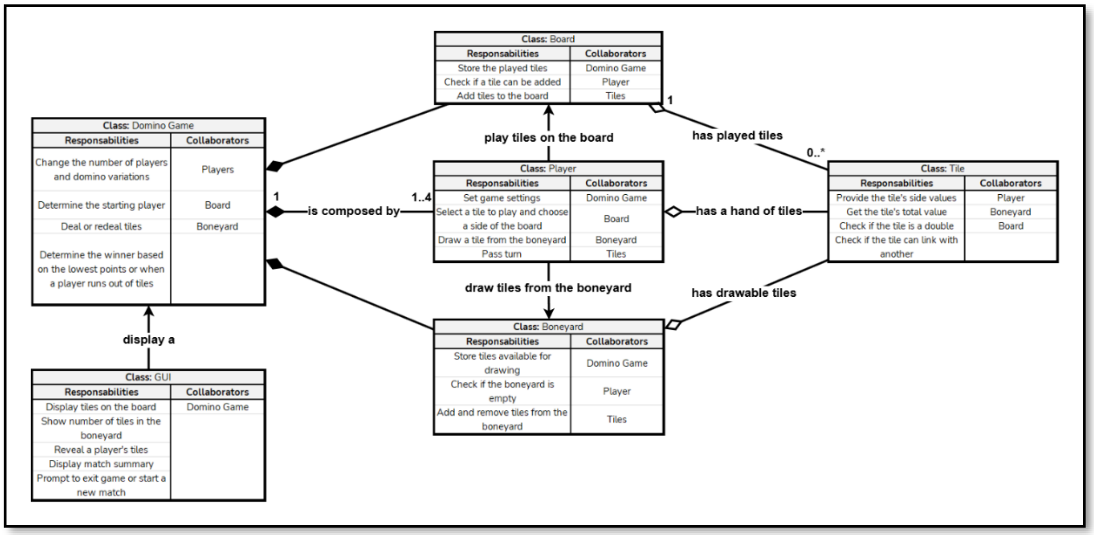
Preconditions:

- All players have been playing, and a winner or tie condition is met.

Use-Case Diagram


Player
Set Up the Game
<<include>>
Select a Domino Variation
<<include>>
Choose the Number of Players
<<include>>
System deals Tiles and selects Starting Player
<<extend>>
<<extend>>
Play the Game
<<include>>
Wait for His Turn to Play
<<include>>
Choose a possible Tile to play on the Board
<<extend>>
Play a Tile that matches the Ends of the Board
<<extend>>
<<extend>>
<<extend>>
<<extend>>
All Players pass their Turn
<<extend>>
Pass the Turn to the next Player
<<extend>>
Draw Tiles from the Boneyard until he can play or the Boneyard is Empty
The Player uses all his Tiles and is declared the Winner
<<include>>
<<include>>
Select the Player with the fewest points as the Winner
<<include>>
End the Game
<<extend>>
Exit the Game
<<extend>>

CRC Cards



Class: Domino Game	
Responsabilities	Collaborators
Change the number of players and domino variations	Players
Determine the starting player	Board
Deal or redeal tiles	Boneyard
Determine the winner based on the lowest points or when a player runs out of tiles	

Class: Board	
Responsabilities	Collaborators
Store the played tiles	Domino Game
Check if a tile can be added	Player
Add tiles to the board	Tiles

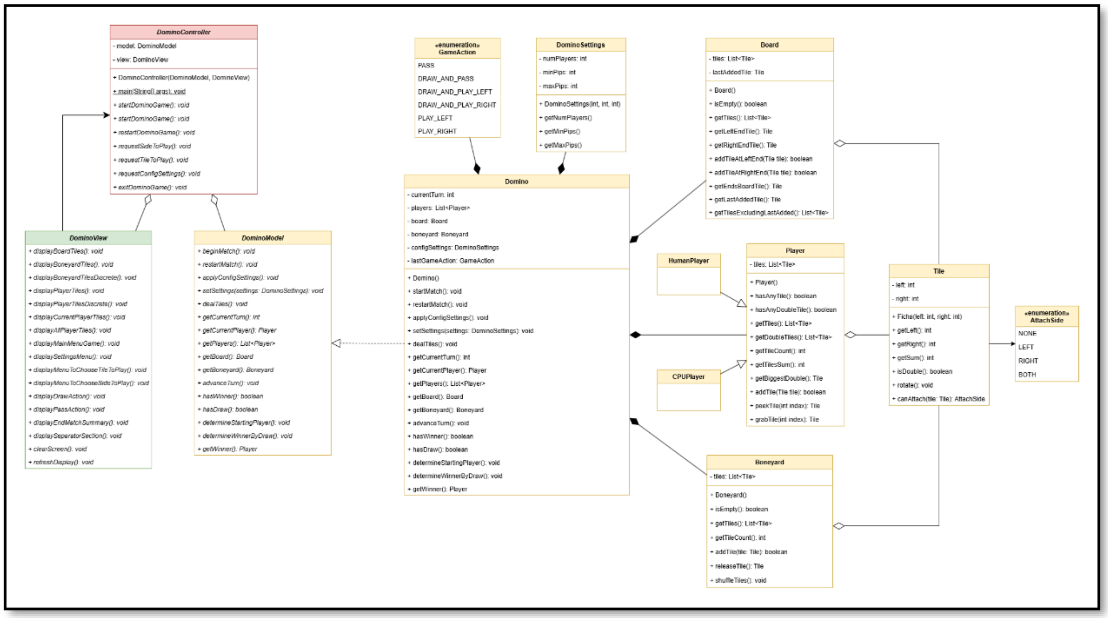
Class: Player	
Responsabilities	Collaborators
Set game settings	Domino Game
Select a tile to play and choose a side of the board	Board
Draw a tile from the boneyard	Boneyard
Pass turn	Tiles

Class: Tile	
Responsabilities	Collaborators
Provide the tile's side values	Player
Get the tile's total value	Boneyard
Check if the tile is a double	Board
Check if the tile can link with another	

Class: Boneyard	
Responsabilities	Collaborators
Store tiles available for drawing	Domino Game
Check if the boneyard is empty	Player
Add and remove tiles from the boneyard	Tiles

Class: GUI	
Responsabilities	Collaborators
Display tiles on the board	Domino Game
Show number of tiles in the boneyard	
Reveal a player's tiles	
Display match summary	
Prompt to exit game or start a new match	

Class Diagram


DominoController
- model: DominoModel
- view: DominoView
+ DominoController(DominoModel, DominoView)
+ main(String[] args): void
+ startDominoGame(): void
+ startDominoGame(): void
+ restartDominoGame(): void
+ requestSideToPlay(): void
+ requestTileToPlay(): void
+ requestConfigSettings(): void
+ exitDominoGame(): void
DominoView
+ displayBoardTiles(): void
+ displayBoneyardTiles(): void
+ displayBoneyardTilesDiscrete(): void
+ displayPlayerTiles(): void
+ displayPlayerTilesDiscrete(): void
+ displayCurrentPlayerTiles(): void
+ displayAllPlayerTiles(): void
+ displayMainMenuGame(): void
+ displaySettingsMenu(): void
+ displayMenuToChooseTileToPlay(): void
+ displayMenuToChooseSideToPlay(): void
+ displayDrawAction(): void
+ displayPassAction(): void
+ displayEndMatchSummary(): void
+ displaySeperatorSection(): void
+ clearScreen(): void
+ refreshDisplay(): void
DominoModel
+ beginMatch(): void
+ restartMatch(): void
+ applyConfigSettings(): void
+ setSettings(settings: DominoSettings): void
+ dealTiles(): void
+ getCurrentTurn(): int
+ getCurrentPlayer(): Player
+ getPlayers(): List<Player>
+ getBoard(): Board
+ getBoneyard(): Boneyard
+ advanceTurn(): void
+ hasWinner(): boolean
+ hasDraw(): boolean
+ determineStartingPlayer(): void
+ determineWinnerByDraw(): void
+ getWinner(): Player
«enumeration»
GameAction
PASS
DRAW_AND_PASS
DRAW_AND_PLAY_LEFT
DRAW_AND_PLAY_RIGHT
PLAY_LEFT
PLAY_RIGHT
DominoSettings
- numPlayers: int
- minPips: int
- maxPips: int
+ DominoSettings(int, int, int)
+ getNumPlayers()
+ getMinPips()
+ getMaxPips()
Domino
- currentTurn: int
- players: List<Player>
- board: Board
- boneyard: Boneyard
- configSettings: DominoSettings
- lastGameAction: GameAction
+ Domino()
+ startMatch(): void
+ restartMatch(): void
+ applyConfigSettings(): void
+ setSettings(settings: DominoSettings): void
+ dealTiles(): void
+ getCurrentTurn(): int
+ getCurrentPlayer(): Player
+ getPlayers(): List<Player>
+ getBoard(): Board
+ getBoneyard(): Boneyard
+ advanceTurn(): void
+ hasWinner(): boolean
+ hasDraw(): boolean
+ determineStartingPlayer(): void
+ determineWinnerByDraw(): void
+ getWinner(): Player
Board
- tiles: List<Tile>
- lastAddedTile: Tile
+ Board()
+ isEmpty(): boolean
+ getTiles(): List<Tile>
+ getLeftEndTile(): Tile
+ getRightEndTile(): Tile
+ addTileAtLeftEnd(Tile tile): boolean
+ addTileAtRightEnd(Tile tile): boolean
+ getEndsBoardTile(): Tile
+ getLastAddedTile(): Tile
+ getTilesExcludingLastAdded(): List<Tile>
HumanPlayer
CPUPlayer
Player
- tiles: List<Tile>
+ Player()
+ hasAnyTile(): boolean
+ hasAnyDoubleTile(): boolean
+ getTiles(): List<Tile>
+ getDoubleTiles(): List<Tile>
+ getTileCount(): int
+ getTilesSum(): int
+ getBiggestDouble(): Tile
+ addTile(Tile tile): boolean
+ peekTile(int index): Tile
+ grabTile(int index): Tile
Tile
- left: int
- right: int
+ Ficha(left: int, right: int)
+ getLeft(): int
+ getRight(): int
+ getSum(): int
+ isDouble(): boolean
+ rotate(): void
+ canAttach(tile: Tile): AttachSide
«enumeration»
AttachSide
NONE
LEFT
RIGHT
BOTH
Boneyard
- tiles: List<Tile>
+ Boneyard()
+ isEmpty(): boolean
+ getTiles(): List<Tile>
+ getTileCount(): int
+ addTile(tile: Tile): boolean
+ releaseTile(): Tile
+ shuffleTiles(): void