DESIGN DOCUMENTATION

Assignment #2

Dots And Boxes

Object-Oriented Software Principles and Design

Group Members:

Chris Mary Benson - U56085268

Nandana Shashi - U05556171

Current Framework:

Our design allows for scalability and reusability through the concepts of:

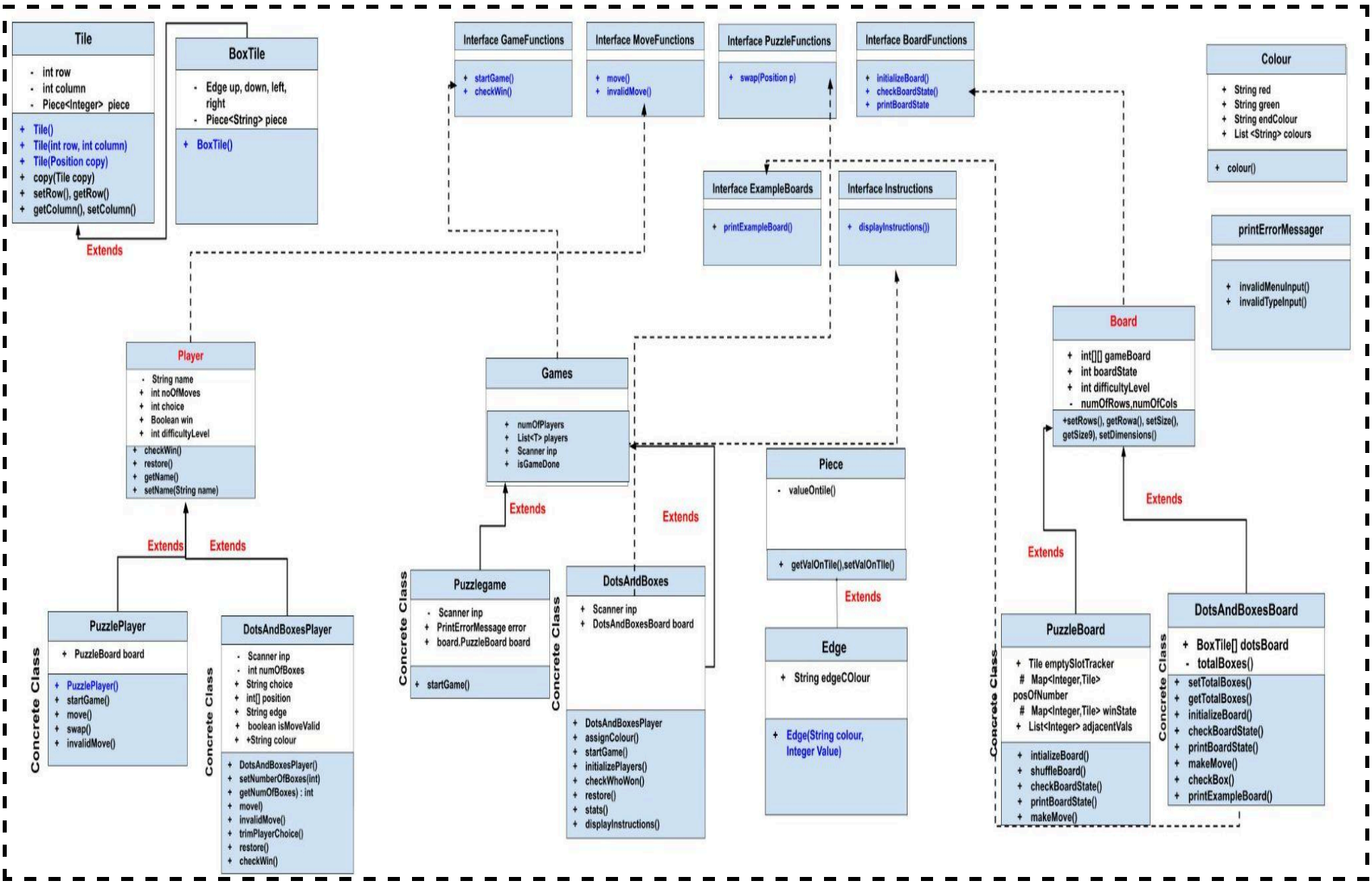
Inheritance:

Inheritance allows for code reusability by having common functions that can be used by all of its child classes. For example, in our framework, we have the dot and boxes game class as well as the sliding puzzle, which extends from the main Games class. Methods such as initializing players, which are common among these games, are declared as an abstract method in the Games class, from which the child classes provide their own implementation.

Further, the Games class also has a list of players of the generic type, which allows for code reusability; methods for retrieving details such as names from the players of a game do not need to be written in each class. A single method in the Games class achieves this with the concept of Generics.

Interfaces

Interfaces enforce reusability, as there can be multiple implementations of the same interface, and at the same time, they enforce extensibility too, as there can be new implementations as well.
In our design, we have interfaces that are implemented by the game, board, and player classes. For example, we have a move interface, which is implemented by the player class; now, all the child classes of player will be able to implement their own logic for the move() method.


Tile
- int row
- int column
- Piece<Integer> piece
+ Tile()
+ Tile(int row, int column)
+ Tile(Position copy)
+ copy(Tile copy)
+ setRow(), getRow()
+ getColumn(), setColumn()
BoxTile
- Edge up, down, left, right
- Piece<String> piece
+ BoxTile()
Extends
Interface GameFunctions
+ startGame()
+ checkWin()
Interface MoveFunctions
+ move()
+ invalidMove()
Interface PuzzleFunctions
+ swap(Position p)
Interface BoardFunctions
+ initializeBoard()
+ checkBoardState()
+ printBoardState
Colour
+ String red
+ String green
+ String endColour
+ List <String> colours
+ colour()
Interface ExampleBoards
+ printExampleBoard()
Interface Instructions
+ displayInstructions())
printErrorMessager
+ invalidMenuInput()
+ invalidTypeInput()
Player
- String name
+ int noOfMoves
+ int choice
+ Boolean win
+ int difficultyLevel
+ checkWin()
+ restore()
+ getName()
+ setName(String name)
Games
+ numOfPlayers
+ List<T> players
+ Scanner inp
+ isGameDone
Board
+ int[][] gameBoard
+ int boardState
+ int difficultyLevel
- numOfRows,numOfCols
+setRows(), getRowa(), setSize(), getSize9), setDimensions()
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Piece
- valueOntile()
+ getValOnTile(),setValOnTile()
Extends
Edge
+ String edgeCOlour
+ Edge(String colour, Integer Value)
Concrete Class
PuzzlePlayer
+ PuzzleBoard board
+ PuzzlePlayer()
+ startGame()
+ move()
+ swap()
+ invalidMove()
Concrete Class
DotsAndBoxesPlayer
- Scanner inp
- int numOfBoxes
+ String choice
+ int[] position
+ String edge
+ boolean isMoveValid
+ +String colour
+ DotsAndBoxesPlayer()
+ setNumberOfBoxes(int)
+ getNumOfBoxes) : int
+ move()
+ invalidMove()
+ trimPlayerChoice()
+ restore()
+ checkWin()
Concrete Class
Puzzlegame
- Scanner inp
+ PrintErrorMessage error
+ board.PuzzleBoard board
+ startGame()
Concrete Class
DotsAndBoxes
+ Scanner inp
+ DotsAndBoxesBoard board
+ DotsAndBoxesPlayer
+ assignColour()
+ startGame()
+ initializePlayers()
+ checkWhoWon()
+ restore()
+ stats()
+ displayInstructions()
Concrete Class
PuzzleBoard
+ Tile emptySlotTracker
# Map<Integer,Tile> posOfNumber
# Map<Integer,Tile> winState
+ List<Integer> adjacentVals
+ intializeBoard()
+ shuffleBoard()
+ checkBoardState()
+ printBoardState()
+ makeMove()
Concrete Class
DotsAndBoxesBoard
+ BoxTile[] dotsBoard
- totalBoxes()
+ setTotalBoxes()
+ getTotalBoxes()
+ initializeBoard()
+ checkBoardState()
+ printBoardState()
+ makeMove()
+ checkBox()
+ printExampleBoard()

Changes from the Previous Framework

1. Introduced Tile and Piece Classes:

The new framework introduces a Tile and a Piece class to represent each tile on the board and its corresponding value, respectively. Each Tile object holds its position on the board and contains a Piece object representing its value. The Piece class holds the content or symbol placed within a tile.

In the Dots and Boxes framework, the Tile class defines positions and relationships for the edges through its child class, BoxTile. BoxTile has an Edge member, which is a subclass of the Piece class.

2. Added a GameStarter Class:

The Game class serves as the main entry point of the program. It creates an instance of the GameStarter class, which is responsible for displaying the game menu and initiating the selected game.

This class handles the display of the main game menu interface. It allows players to choose which game to play, making the framework more interactive and organized by separating menu-related functionality from the main game logic.

3. Introduced a Central Game Class:

A central Games class has been implemented to act as the core controller/parent of the system. It manages shared functionalities such as initializing players, setting up the board, and handling the main game loop. From this Games class, the specific game classes for each individual game inherit from it, allowing shared functionality while maintaining separate logic for each type of game. This improves code reusability and maintainability, ensuring consistency across different games.

The Games class contains a List of a generic type to store the players of any type of game and to initialize those players and get their details. This ensures code reusability as well as code extendability, as this can be used in multiplayer games.

4. Created Specific Game Classes:

Specific game classes have been introduced for each game type. The Dots and Boxes game and the Sliding Puzzle game now each have their own dedicated classes that extend the central Game class. This ensures that each game can implement its unique rules and gameplay mechanics while still following a unified structure within the framework.