# Design Documentation

# High Level Architecture



**Figure 5.1.1: High-Level UML Interaction Diagram**

This is a description of the high-level architecture and class interactions of the program. More detailed UML diagrams, class structures, interactions, method descriptions and requirement references for the methods are included in the following sections. The source code includes pseudo-code incorporated into every method. The pseudo-code is included with each line of the code using // comments.

The program begins with the Game class, which contains the main() method and one GUI object. The GUI class is how the user interacts with the game. It has buttons that may be used to perform the various events. Each button has its own GUIHandler, which are nested inner classes of the GUI and implement the GUI and implement the ActionListener interface. These handlers use attributes from both the Board class and the PlayerManager class. These buttons also purchase and play objects extended from the DevelopmentCard abstract class and roll the Dice object. Thus, the GUI has attributes which include one Board object, one PlayerManager object, the deck of DevelopmentCards, which contains 25 cards, and the Dice object.

The Board object displays the Tile objects, which are contained as a linked list in the TileList object. Each Tile has a Location object. The Board also has two LocationList objects. These are linked lists of Location objects. One list for Road Locations and one list for Settlement/City Locations.

Location objects are designed to hold GamePiece objects. The Locations in the Tiles are designed so the Robber may be moved to the different Tiles. The LocationLists are designed to hold GamePiece objects which belong to the player, which are Road objects, Settlement object and City objects.

The game is designed to scale to the native resolution of the monitor. Visible components, such as the JFrames, JComponents, the GUI, Tile images and certain text will scale so that the game can be played on a variety of monitor resolutions. The scale is set in the Game class.

# Game Class



**Figure 5.2.1**

* Game class requirements: R1.5.0, R2.0.0, R4.0.0, R4.1.0 – 4.1.5, R5.4.2

The Game class creates a JFrame to hold other components, sets the scale, and is responsible for the set-up phase of the game. The game was designed around a height of 1000 pixels. The scale is based off the ratio of the screen’s native resolution to 1000. The width of the JFrame is set to the width of the native resolution.

The set up-phase is when each player takes turns setting up two Settlements with adjacent Roads and is accomplished with the methods placeInitialSettlements() and setInitialSettlements(). It also includes two static overloaded methods, which are called showSelectionBox(). These methods are used at various points in the program when the player must choose an item from a list or an array.

# GUI Class



**Figure 5.3.1 – The GUI Class Diagram with Nested Inner Classes.**

* GUI class requirements: R2.0.0 through R2.4.6, R3.1.3, R5.1.0, R5.2.4, R7.1.0 – R7.1.5, R7.8.0
* GUIHandler: R2.0.0
* RoadHandler, SettlementHandler, CityHandler: R3.2.0
* BuyDevelopmentCardHandler: R7.3.0
* PlayDevelopmentCardHandler: R7.2.0
* TradeHandler: R8.0.0
* EndTurnHandler: R2.4.5

The GUI extends JPanel and has seven buttons, each with its own nested Handler class. The GUIHandler implements the ActionListener interface and abstracts the commonalities of each button. Each Handler extends the GUIHandler and implements the differences in the buttons in a method called performNextEvent(). The use cases are diagramed in figure 6.3.2. Other than the Handler called and LocationList used, the buttons which build GamePieces have the same sequence, which is diagramed in figure 6.3.3. The Build Development Card button simply adds a DevelopmentCard from the deck to the Player’s DevelopmentCard list. The Play Development Card button displays a selection box if the Player has any cards. If the player selects one, the playCard() method is called and the card is activated. The sequence diagram is shown in figure 6.3.4. The Trade button displays a selection box, which allows the user to select any number of their resources to trade, then a selection box is displayed, which allows the user to request any number of resources. For every Player in the game, starting with the first, if a Player has the requested resources, a trade is offered. The player may accept or reject the trade offer. The End Turn button checks with the PlayerManager to see if the player has earned the largest army, or if the Player has won the game. If the game is not over, the next Player becomes the current Player and the Dice are rolled for the new player.



**Figure 5.3.2 – Use Case Diagram for GUI Buttons**



**Figure 5.3.3 - Sequence Diagram for Building Game Pieces (Road, Settlement, City)**



**Figure 5.3.4 – Sequence Diagram for Playing Development Card**

# Board Class



**Figure 5.4.1 – The Board class Diagram**

* Board requirements: R1.0.0, R1.4.0, R3.0.0, R4.1.1, R4.1.3, R4.1.5, R5.2.1, R5.2.3, R5.2.4, R6.0.0 – R6.4.0, R7.4.0

The Board class is responsible for displaying the Tiles and GamePieces. It also pays resources to Players from the Tiles during the setup phase and after Dice rolls. It displays LocationButtons when the user requests to build a Road, Settlement, or City and they have the proper resources available. It also displays LocationButtons when the Robber is to be moved. GamePieces are placed by clicking the Location Buttons.

# Tile & Associated Classes



**Figure 5.5.1 – TileList Build with Associations**

* TileList class requirements: R1.0.0, R1.2.0 – 1.2.7
* Tile class requirements: R1.0.0, R1.2.0 – R1.2.7, R1.3.2, R1.3.3, R5.2.1
* Resource enumeration: R1.2.1 – 1.25
* NumberTokens: R1.3.0, R1.3.4
* NumberToken: R1.3.1, R1.3.2, R5.2.1
* Value enumeration: R1.3.1

The TileList object is a linked list of Tile objects. Each Tile has a Resource enumeration associated with it. Each Tile also has a NumberToken. Each NumberToken has a Value enumeration associated with it. Each NumberToken is built into the NumberTokens List so they may be shuffled and easily distributed to the tiles. The Tiles are stored in the TileList for the same reasons. The Tiles are built, then added to the The TileList is added to the Board to create the tiles.

# Location, LocationList, LocationButton Classes



**Figure 5.6.1 – LocationList class with Location class**

* Location class requirements: R3.2.2, R3.2.3, R5.2.1, R5.2.3, R6.0.0
* LocationList class requirements: R3.2.2, R3.2.3

The Location class is designed to hold GamePiece objects. They are in Tile centers so the

Robber can be placed in any Tile. For Roads, Settlements and Cities the Location objects are placed in LocationLists for iterative purposes. Two LocationLists are created. One to hold Roads and one to hold Settlements and Cities. The Road LocationList places Location objects along the edges of the tiles. The Settlement LocationList, which is used for Settlements and Cities, places Location objects at the corners of the Tiles.



**Figure 5.6.2 – LocationButton class**

* LocationButton class requirements: R6.4.0

LocationButton objects are designed to be placed at a Location objects. When a user requests to build a GamePiece (Road, Settlement or City), LocationButtons are placed on the Board where the GamePiece may be placed. When an event occurs that requires the Robber is to be moved, LocationButtons are placed at the Tile centers. When the user clicks the LocationButton, the Board attribute gamePieceReadyToPlace is placed at the Location of the LocationButton.

# Player and PlayerManager Classes



**Figure 5.7.1 – Player class, PlayerManager class and PlayerColor Enumeration**

* Player class requirements: R1.1.0 – R1.1.2, R2.2.0 – R2.2.2, R3.1.0 – R3.1.3, R4.1.5, R5.2.1, R5.2.2, R5.2.4, R6.1.0 – R6.1.2, R7.2.0 – R7.6.0, R8.1.0, R9.0.0, R9.5.0, R9.5.1
* PlayerManager class requirements: R1.1.0 – R1.1.2, R2.1.0 – R2.2.1, R2.4.5, R4.1.0, R5.2.2, R7.5.0, R8.1.0, R9.5.0, R9.5.1, R9.6.0
* PlayerColor enumeration requirements: R1.1.0 – R1.1.2

The Player class is responsible for creating GamePieces for the Player and keeping track of the Player’s GamePieces, Resources, DevelopmentCards, and score. It has a string representation of the Player’s information to display in the GUI.

PlayerManager class is responsible for creating and keeping track of the Player’s in the game. It contains the PlayerColor enumeration, which is used to differentiate the Players. The color of the Player’s GamePieces match the name enumeration name. The PlayerColor enumeration also has a Color attribute, which is used to set the color of the GUI for each Player during their turn.

# GamePiece Abstract Class with Subclasses



**Figure 5.8.1 – GamPiece abstract class with Robber, Road, Settlement, and City subclasses**

* GamePiece requirements: R3.0.0, R5.2.1, R6.0.0
* Robber requirements: R1.4.0, R5.2.3, R7.4.0
* Road requirements: R3.1.2, R3.2.1, R3.2.3
* Settlement requirements: R3.1.0, R3.2.1, R5.2.1, R9.1.0
* City requirements: R3.1.1, R3.2.1, R5.2.1, R9.2.0

The GamePiece subclasses create objects which are game pieces that may be placed on the Board. They are placed in Location objects using LocationButtons. The Robber can be placed at Tile centers. Roads can be placed in one LocationList, and Settlements and Cities may be placed in the other LocationList. Each GamePiece must override the place(Location) method, which adds them to the Board at the appropriate Location.

# DevelopmentCard Abstract Class with Subclasses



**Figure 5.9.1 – DevelopmentCard abstract class with subclasses.**

* DevelopmentCard requirements: R7.0.0, R7.2.0, 7.3.1
* KnightCard requirements: R7.1.1, R7.2.0, R7.4.0
* MonopolyCard requirements: R7.1.4, R7.2.0, R7.5.0
* RoadBuildingCard requirements: R7.1.3, R7.2.0, R7.8.0
* VictoryPointCard requirements: R7.1.2, R7.2.0, R7.7.0
* YearOfPlentyCard requirements: R7.1.5, R7.2.0, R7.6.0

The DevelopmentCard subclasses create the development cards. Each subclass must override the playCard() method, which allows the Player to receive the benefits. A toSring() method, must also be overridden.

# Dice Class

 

**Figure 5.10.1 – Dice class Figure 5.10.2 – Value Enumeration in NumberTokens class**

* Dice requirements: R5.0.0, R5.1.0
* Value enumeration requirements: R5.2.1

The Dice class returns two randomly generated numbers between 1 and 6 to simulate rolling two dice. Two dice must be rolled to correctly represent the odds for each roll. A NumberTokens.Value Enumeration is returned from the getValue() method, which depends on the roll. This Enumeration is checked against each Tile’s NumberToken value. When they match, the Tile pays appropriate resources to each Settlement or City at each Location on the corners of the Tile.