Team 3 Architecture Document

This document contains a discussion on design choices made for milestone 2 and 3 in our Settlers of Catan project. The original UML from the project and a partially connected UML for milestone 3 are included along with a writeup. We recommend viewing the UML in browser at the following link:

https://github.com/yilmazaj/CSSE375Project/tree/main/SoftwareArtifacts

They are also available for viewing at the bottom of the document in much lower quality.

Main Design Decisions for Each Feature

Milestone 3

Feature 1. Added GUI interaction for road placement

Originally the game required users to input text into JOption input panels to be able to place structures on the Settlers of Catan board, this was clunky and unintuitive. The goal of this feature was to replace the input panels with being able to click on the board display itself in order to select where to place the structures. We were successfully able to accomplish this and now users can easily select where to build roads, settlements, and other structures by clicking on the buttons that appear above each intersection when it is time to build a structure.

Feature 2. Revamped GUI interaction for trade

The game initially required users to go through a lengthy, and annoying process of trading. Each time a player wanted to trade, they would have to enter resource amounts in individual JOptionPane popups. Then, they would have to type the name of the player they wanted to trade with, prompting another JOptionPane for confirmation, etc. After the trade would complete, the system would prompt the user to trade again, ie: a loop until the player decided not to. The goal of this feature was basically to overhaul the entire look, feel, and experience of the trading system. Now, users input all resource amounts to give away on the same UI. Similarly, users input all resource amounts they wish to receive on the same UI. Then, the system analyzes the requested resources and displays each player's name on a "Trade With" screen, only allowing players with the requested resource amounts to accept the trade. If all players deny the trade, the trade ends. Overall, this feature has revamped the trading subsystem to be far more user friendly by making the trading process quicker and clearer for all players.

Feature 3. Added GUI interaction for moving the robber

All the original robber movement had to happen through a series of JFrames that took the user input to select which hex to move the robber. This required the players having knowledge of the hex numbering in order to input a valid hex to move the robber. Occasionally, players would count incorrectly and move the robber to a hex adjacent to the intended one. My feature remedies these issues by displaying buttons on each hex for the players to select when moving the robber. This solves the issue of invalid hex number inputs, because the buttons are already corresponding to a valid hex. It also reduces the cognitive strain on players attempting to figure out the hex numbering as they can just select the hex from the game board by clicking on it. The main decision I made was to separate functionality into HexButtonManager and HexButtonActionListener classes.

Feature 4. Design changes to resolve god class

In the original codebase, the Game class served the purpose of a sort of god class, controlling everything related to the game's flow, player actions, structure placement, etc. The purpose of this feature is to redistribute some of the responsibility out of the Game class to help lighten its workload, improve its cohesion, and reduce its size. The distribution that was decided on was to split off the functionality that relied on the GameBoard object into its own class. Incidentally, all of these functions were related to building placement and creation. These functions were put into a new GameBuildingHandler class. It was decided that the functions not go into the GameBoard object itself since the moved functions mostly relied on use of a Player as well, which sufficient reason to leave it separate from both GameBoard and Game.

Milestone 2

Feature 1. Added GUI card for dice rolls and player turn

When adding the GUI for player turns and dice rolling, we detached the simple random number generator from the game class and made it into its own Dice class. We could then pass a copy of this to the separate GUI class which allowed us to separate the classes a bit more as they didn't have to share the dice roll information directly. This new GUI class replacing the GUI built into the game class also drastically decreased coupling and gave us a greater cohesion between the two parts. In the future it should be fairly simple to swap out the GUI with another and allows us to expand upon the GUI easily.

Feature 2. Added GUI for displaying player stats

When adding the GUI for displaying player statistics, we first had to do away with the console logs displaying rolls, resource amounts, and other such related information. To create a new GUI with all of the player details, we had to create an instance of the PlayersStatsGUI Class (playerStats) in the Game class, and pass it the array of Players upon construction. From there, the PlayersStatsGUI creates individual PlayerStatsGUI Class instances for displaying each player's statistics, then compiles them together in a JFrame. From this point, the Game class simply needs to make a one line call to playerStats's method: updatePlayerStats() whenever resource amounts are changed, which iterates through each individual PlayerStatsGUI instance and updates resource amounts. These new GUI classes replace the console logs built into the game, and also keeps GUI responsibilities out of the game class. The single method call to update all GUI's and the flexible iteration creating individual PlayerStatsGUI classes means that making UI changes in the future should require no modification to Game, nor PlayersStatsGUI, and should only require changes to be made at the lowest level (PlayerStatsGUI).

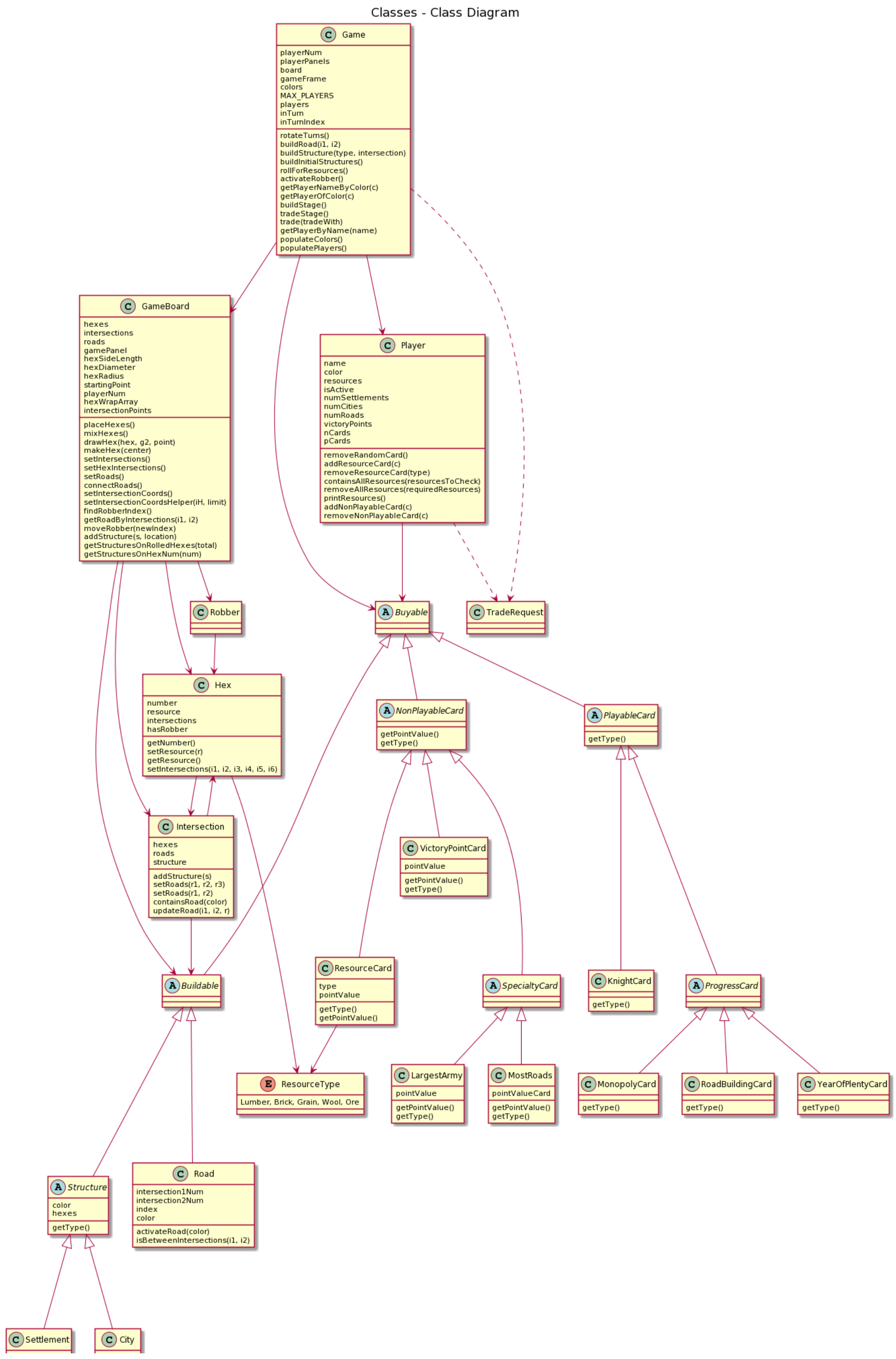
Feature 3. Added and improved icons for buildings and resources

For adding resource icons to the board we decided to use subclasses of an abstract Hex class to separate the different icons. We decided to use the string type of the hex as the icon's image name which allowed us to generalize the icon drawing to the superclass. We decided to draw the icons in the drawHex method because the resource hex icons only need to be drawn once. For the structure icons, we took a similar approach to resources, though we ultimately ended up collapsing the subclasses to the Structure class. The icon drawing is similar however, with the icon's image file having the structure's string type. We decided to call the structure icon drawing method in the paint component class, which would ensure that the structure icons are redrawn and updated as players build structures.

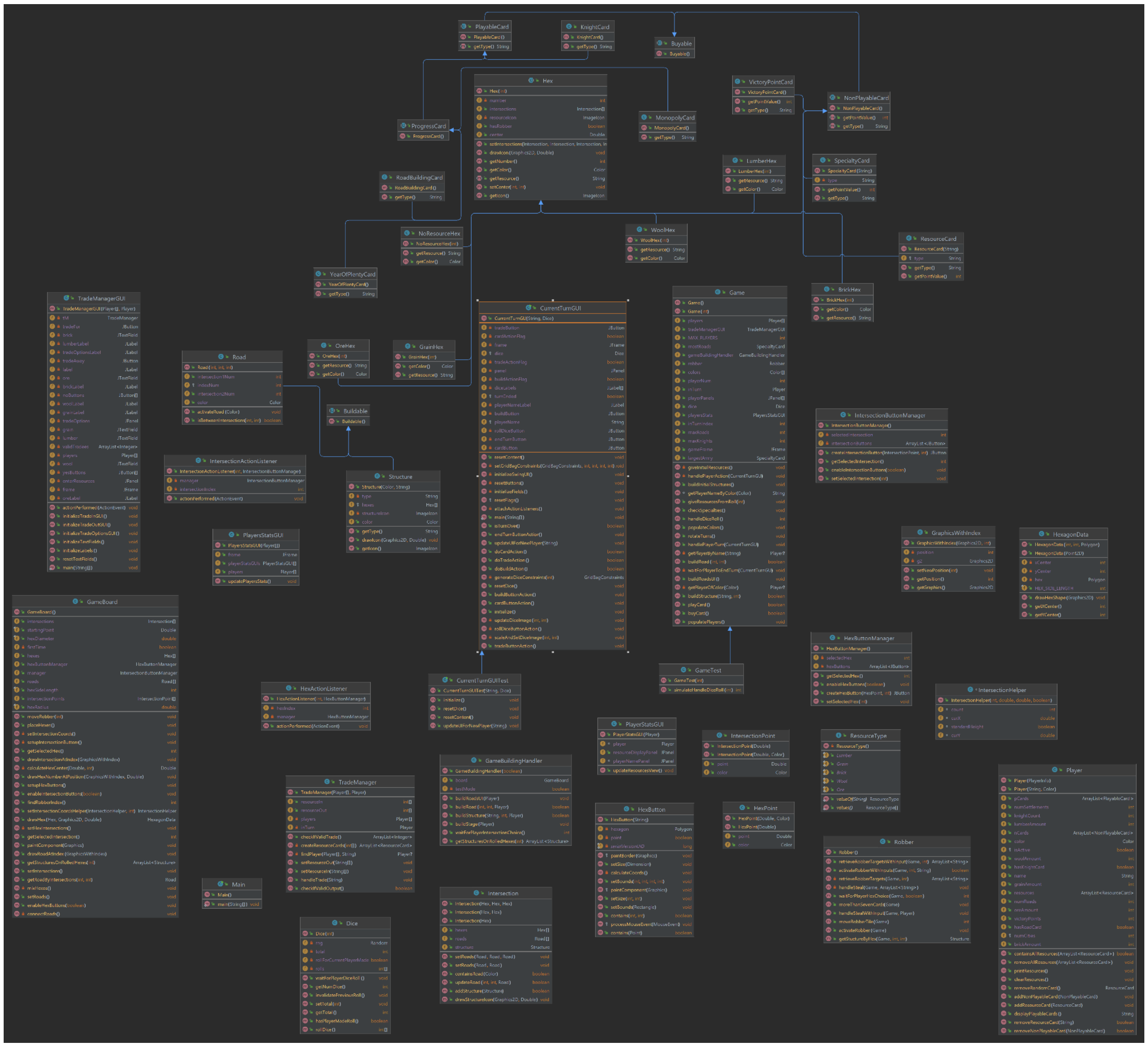
Feature 4. Added implementation for robber

The majority of the design decisions were already made in terms of Robber. There was already a class placeholder for the Robber. The main decision that I made was to move the activateRobber functionality from Game to Robber because it was an instance of feature envy. The activateRobber class already called the required methods, but the methods were not working correctly and certain methods were basically stubs. I filled these out with relevant code to get Robber to function correctly and implemented my own tests to verify. I also verified manually by running the game. That being said, implementing Robber did not involve many major design decisions.

Old UML


Classes - Class Diagram
Game
playerNum
playerPanels
board
gameFrame
colors
MAX_PLAYERS
players
inTurn
inTurnIndex
rotateTurns()
buildRoad(i1, i2)
buildStructure(type, intersection)
buildInitialStructures()
rollForResources()
activateRobber()
getPlayerNameByColor(c)
getPlayerOfColor(c)
buildStage()
tradeStage()
trade(tradeWith)
getPlayerByName(name)
populateColors()
populatePlayers()
GameBoard
hexes
intersections
roads
gamePanel
hexSideLength
hexDiameter
hexRadius
startingPoint
playerNum
hexWrapArray
intersectionPoints
placeHexes()
mixHexes()
drawHex(hex, g2, point)
makeHex(center)
setIntersections()
setHexIntersections()
setRoads()
connectRoads()
setIntersectionCoords()
setIntersectionCoordsHelper(iH, limit)
findRobberIndex()
getRoadByIntersections(i1, i2)
moveRobber(newIndex)
addStructure(s, location)
getStructuresOnRolledHexes(total)
getStructuresOnHexNum(num)
Player
name
color
resources
isActive
numSettlements
numCities
numRoads
victoryPoints
nCards
pCards
removeRandomCard()
addResourceCard(c)
removeResourceCard(type)
containsAllResources(resourcesToCheck)
removeAllResources(requiredResources)
printResources()
addNonPlayableCard(c)
removeNonPlayableCard(c)
Robber
Buyable
TradeRequest
Hex
number
resource
intersections
hasRobber
getNumber()
setResource(r)
getResource()
setIntersections(i1, i2, i3, i4, i5, i6)
NonPlayableCard
getPointValue()
getType()
PlayableCard
getType()
Intersection
hexes
roads
structure
addStructure(s)
setRoads(r1, r2, r3)
setRoads(r1, r2)
containsRoad(color)
updateRoad(i1, i2, r)
VictoryPointCard
pointValue
getPointValue()
getType()
ResourceCard
type
pointValue
getType()
getPointValue()
Buildable
SpecialtyCard
KnightCard
getType()
ProgressCard
ResourceType
Lumber, Brick, Grain, Wool, Ore
LargestArmy
pointValue
getPointValue()
getType()
MostRoads
pointValueCard
getPointValue()
getType()
MonopolyCard
getType()
RoadBuildingCard
getType()
YearOfPlentyCard
getType()
Structure
color
hexes
getType()
Road
intersection1Num
intersection2Num
index
color
activateRoad(color)
isBetweenIntersections(i1, i2)
Settlement
City

Milestone 3 UML


PlayableCard
KnightCard
Buyable
VictoryPointCard
NonPlayableCard
Hex
ProgressCard
MonopolyCard
RoadBuildingCard
LumberHex
SpecialtyCard
NoResourceHex
WoolHex
ResourceCard
YearOfPlentyCard
BrickHex
TradeManagerGUI
CurrentTurnGUI
Game
Road
OreHex
GrainHex
Buildable
IntersectionButtonManager
IntersectionActionListener
Structure
PlayersStatsGUI
GraphicsWithIndex
HexagonData
GameBoard
HexButtonManager
GameTest
CurrentTurnGUITest
HexActionListener
IntersectionHelper
PlayerStatsGUI
IntersectionPoint
ResourceType
Player
GameBuildingHandler
TradeManager
HexButton
HexPoint
Robber
Main
Intersection
Dice