Design Models
Paul's Team

COMP 361 D1: Software Engineering Project
Winter 2015

February 13th, 2014

Julie Morrissey(260526172)
Paul Suddaby (260536057)
Guillaume Labranche (260585371)
Ali Bhojani (260450548)
Andrej Gomizelj (260362874)

Table of Contents

Justification of Classes

New Game

We felt that New Game belonged in the **Game** class because in our design, Game will be in charge of keeping track of aspects of the game state (villages, tiles, active players, etc.) as well as commanding other classes to perform actions (upgrade, move, etc.). As such, it made sense to have the various initializations that happen during the new game method to take place within this class to later be accessed directly for game play.

Begin Turn

We decided the Begin Turn method belonged in the **Game** class, as Game is in charge of keeping a record of Players, Villages, and Tiles, all of which are necessary in the functionality of Begin Turn.

Build Road

The Build Road method was placed in the **Village** class because, in our design, we envisioned Players commanding Units *through* their villages. As in, with the GUI, the Player will command all Units within one village at a given time, and the GUI will announce if any Units in that village are remaining idle for the turn. Given this relationship, it seemed natural that Village command its own Units to manipulate its own Tiles.

Upgrade Village

We felt that the Upgrade Village method best fit in the **Game** class. It makes the most conceptual sense for the Game class to act on and manipulate the state of a Village than for a Village to try and upgrade itself.

Upgrade Unit

Similar to the Build Road method, we felt that Upgrade Unit best belonged in the **Village** class, simply because we envisioned the Players interacting with Villages to give their Units orders. Simply put, Villages are responsible for their own Units.

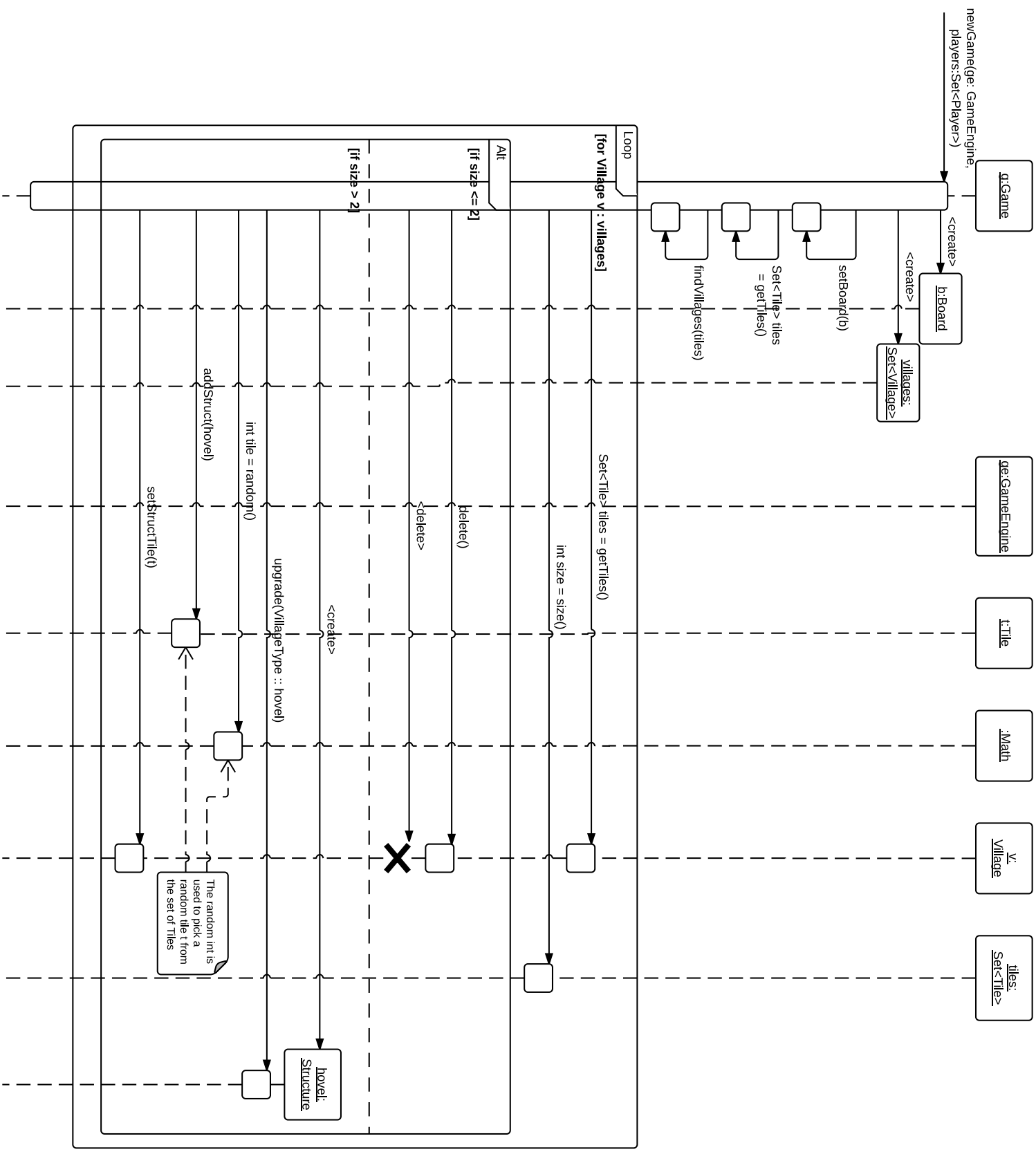
Takeover Tile

To complete the functionality of the Takeover Tile method, our program needs to access to not only the current Player's Villages and Tiles, but also to other Players' information. As such, unlike the other Unit-controlling methods, Takeover Tile was placed in the **Game** class as the Village class would not have a wide enough scope.

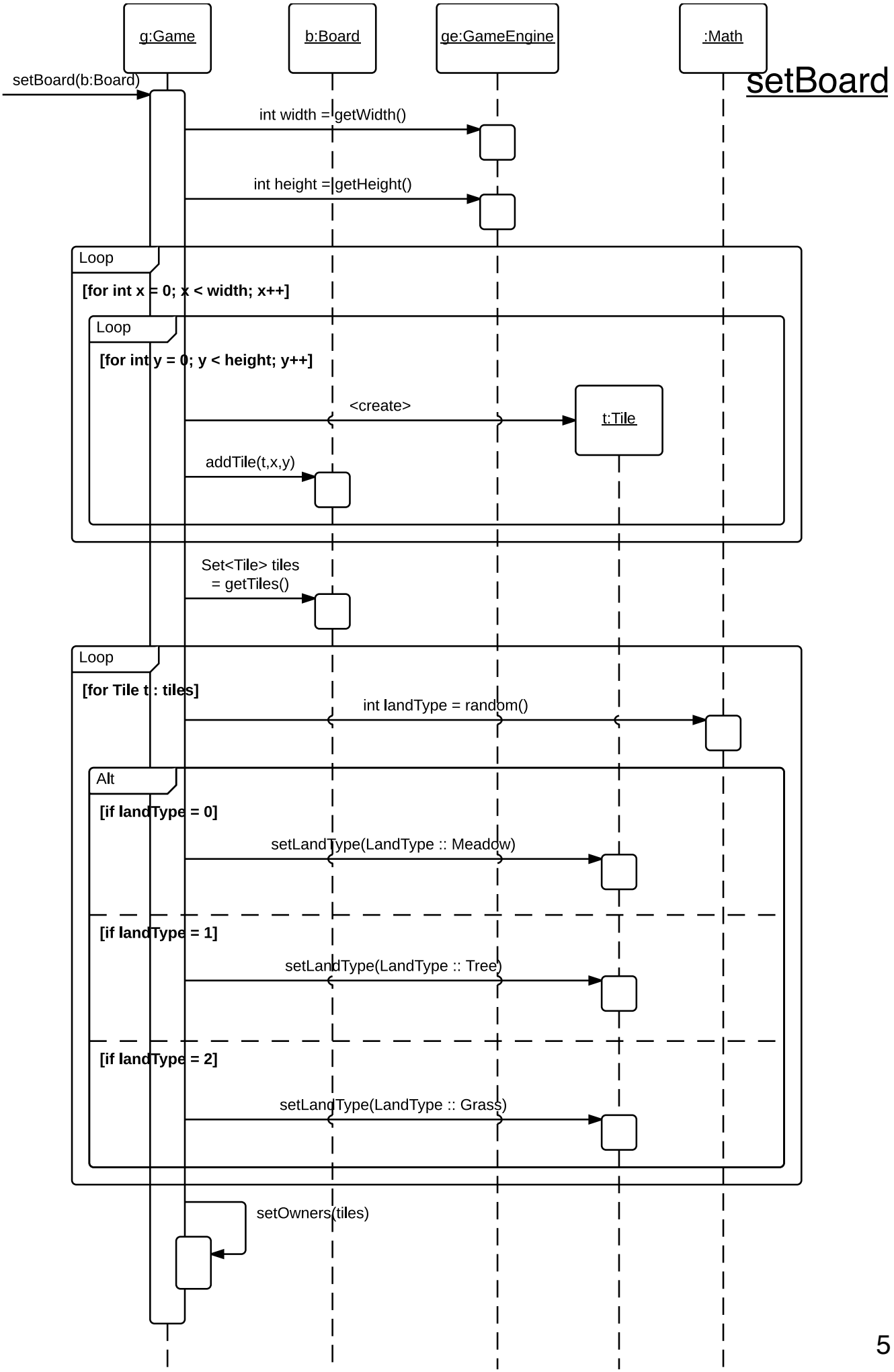
Move Unit

Finally, the Move Unit method belonged in the **Game** class for exactly the same reasons as the Takeover Tile method – the functionality of the method demanded a certain amount of access to other Player's information and if it were not in the Game class, the Game class would be involved as a middle man anyway.

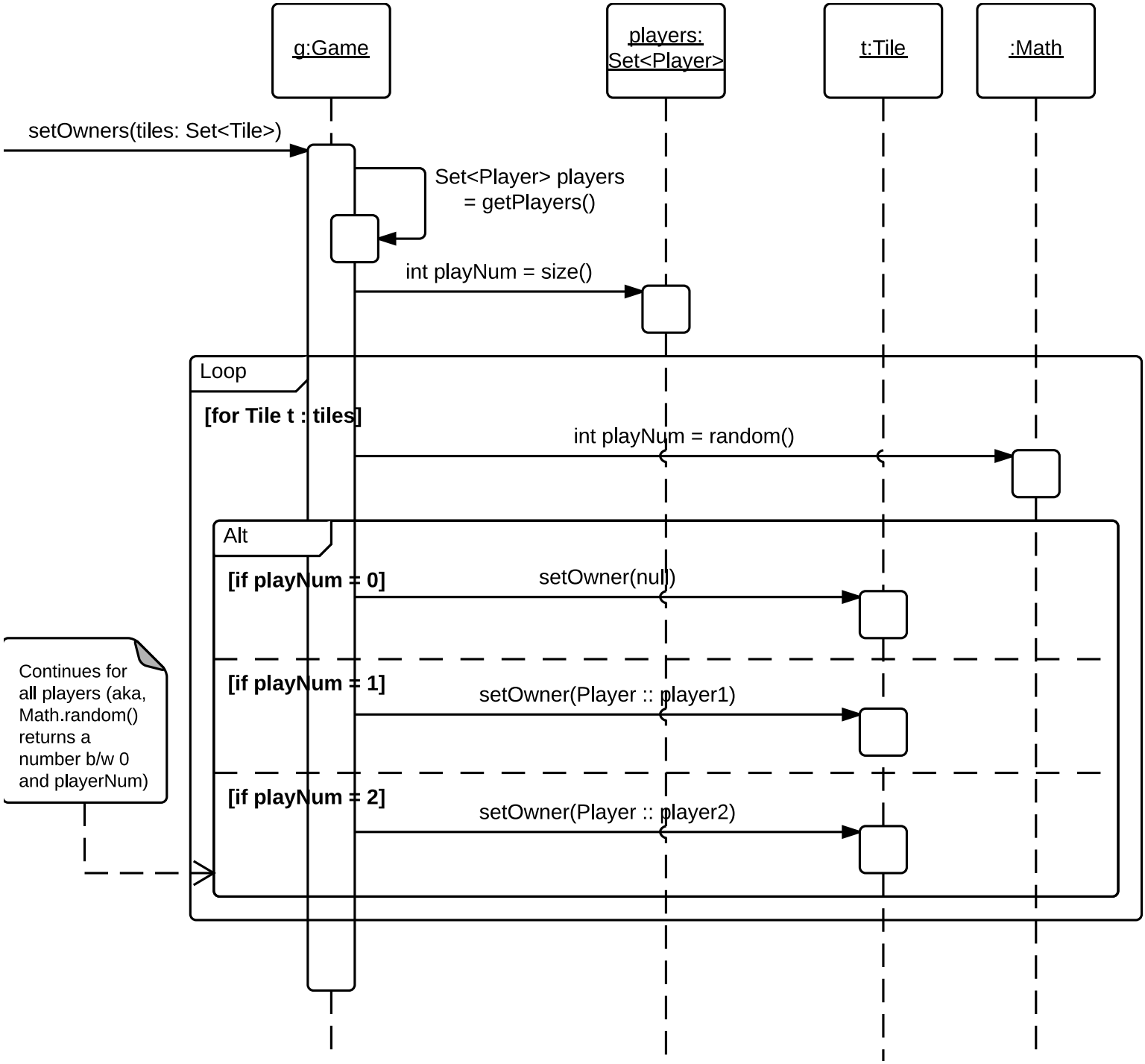
New Game


gr:Game
ge:GameEngine
t:Tile
:Math
v:
Village
tiles:
Set<Tile>
newGame(ge: GameEngine,
players:Set<Player>)
<create>
b:Board
<create>
villages:
Set<Village>
setBoard(b)
Set<Tile> tiles
= getTiles()
findVillages(tiles)
Loop
[for Village v : villages]
Set<Tile> tiles = getTiles()
int size = size()
Alt
[if size <= 2]
delete()
<delete>
[if size > 2]
<create>
upgrade(VillageType :: hovel)
int tile = random()
addStruct(hovel)
setStructTile(t)
The random int is
used to pick a
random tile t from
the set of Tiles
hovel:
Structure

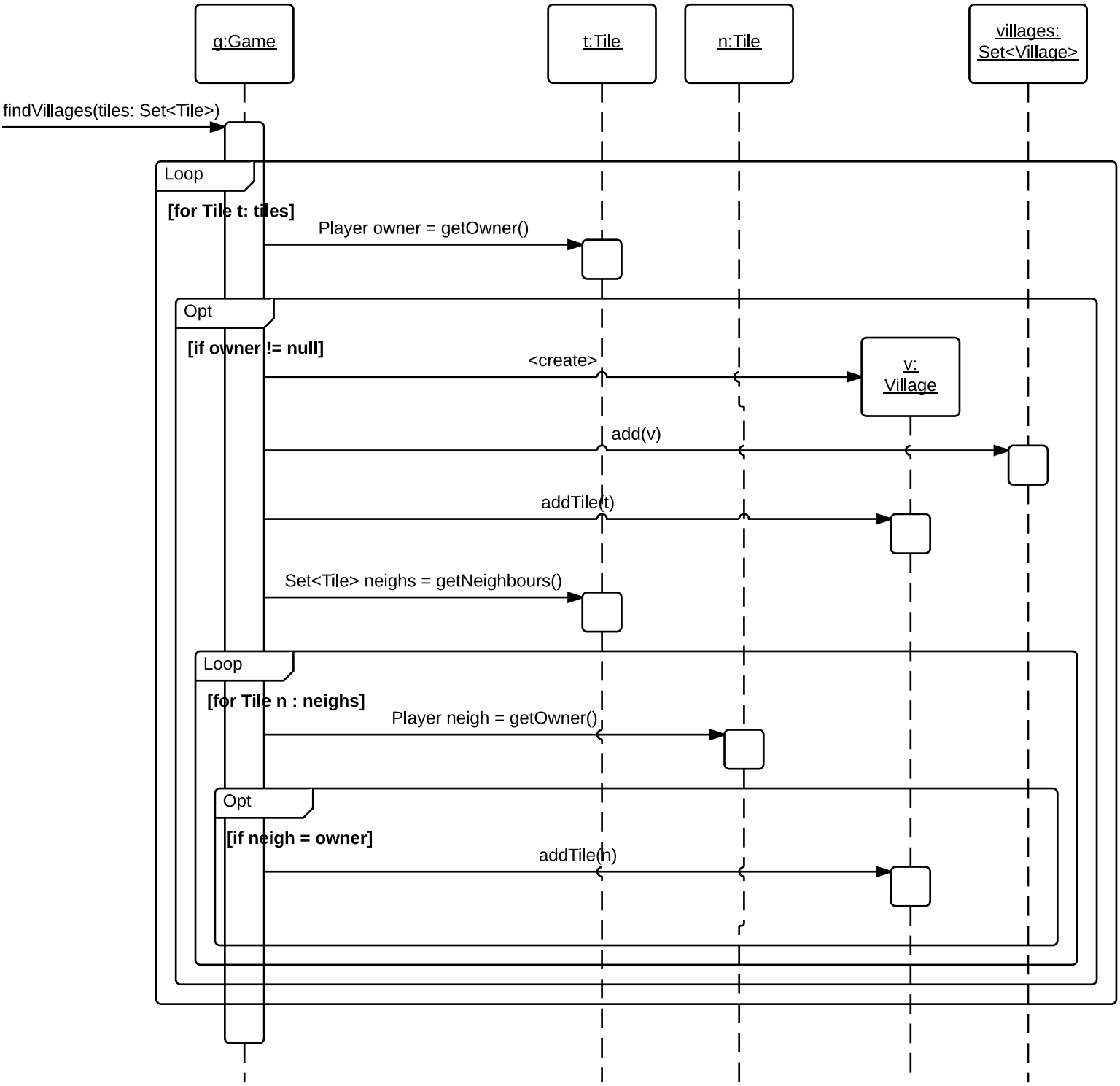
setBoard

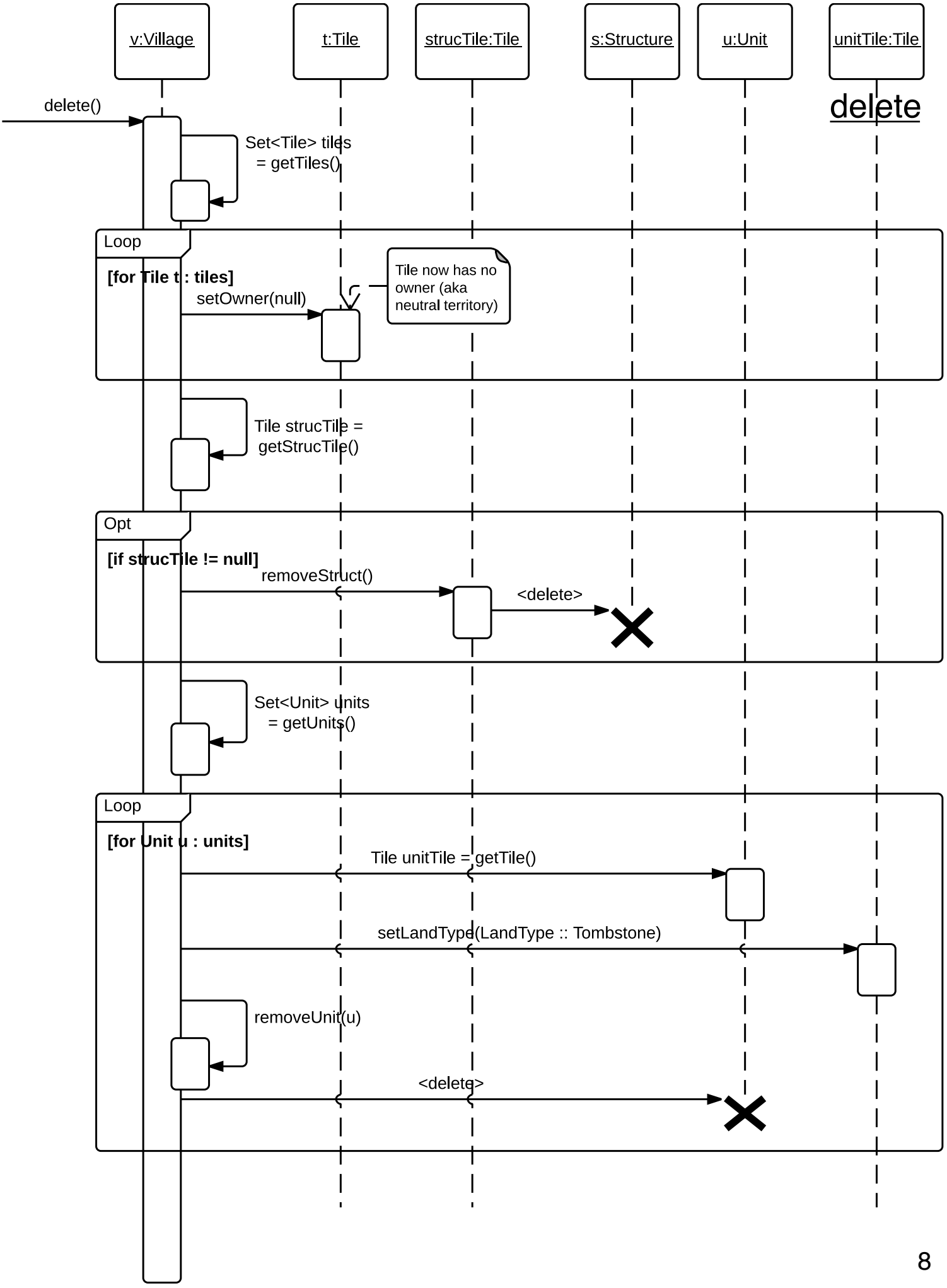

g:Game
b:Board
ge:GameEngine
:Math
setBoard(b:Board)
int width = getWidth()
int height = getHeight()
Loop
[for int x = 0; x < width; x++]
Loop
[for int y = 0; y < height; y++]
<create>
t:Tile
addTile(t,x,y)
Set<Tile> tiles
= getTiles()
Loop
[for Tile t : tiles]
int landType = random()
Alt
[if landType = 0]
setLandType(LandType :: Meadow)
[if landType = 1]
setLandType(LandType :: Tree)
[if landType = 2]
setLandType(LandType :: Grass)
setOwners(tiles)

setOwners

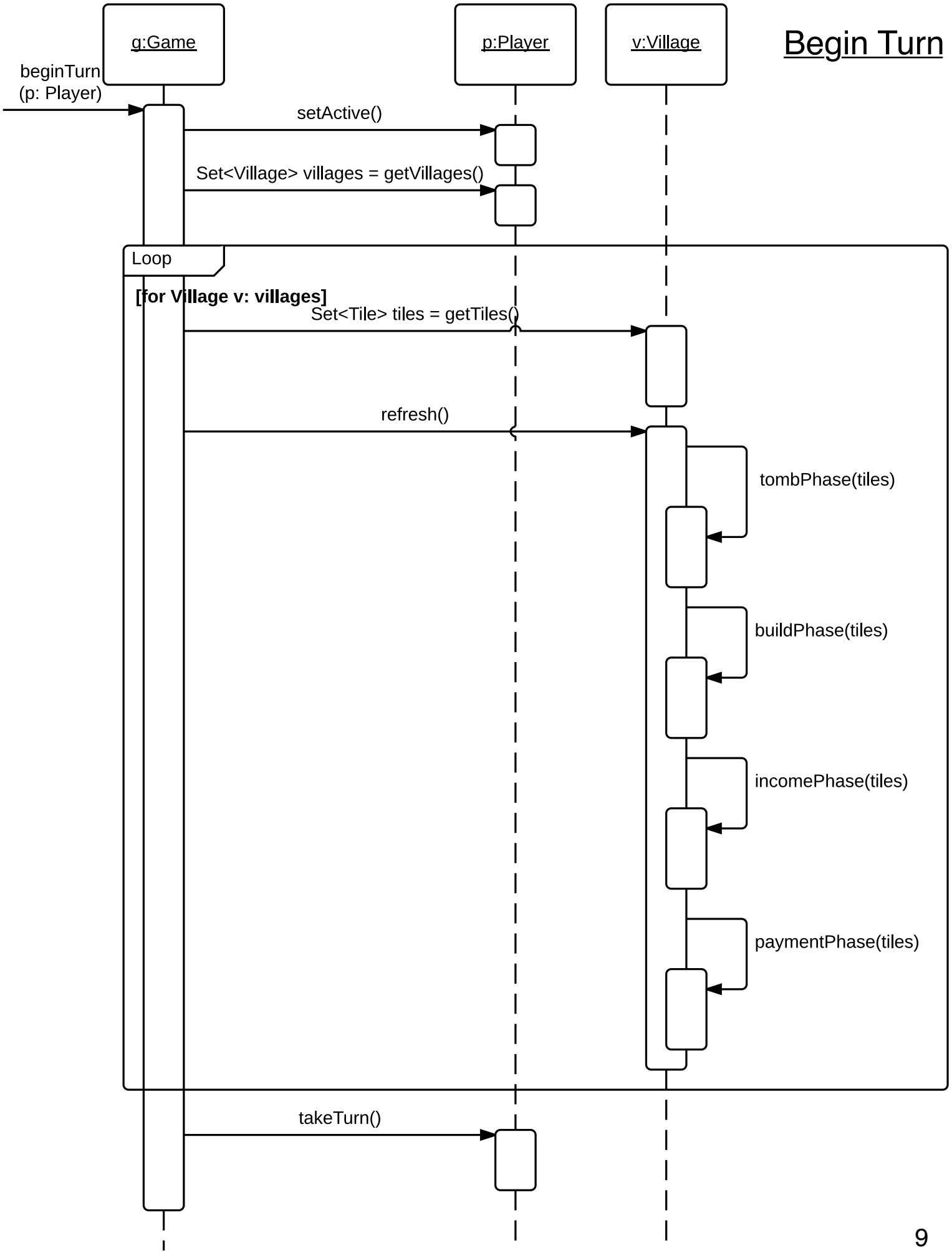

g:Game
players: Set<Player>
t:Tile
:Math
setOwners(tiles: Set<Tile>)
Set<Player> players = getPlayers()
int playNum = size()
Loop
[for Tile t : tiles]
int playNum = random()
Alt
[if playNum = 0]
setOwner(null)
[if playNum = 1]
setOwner(Player :: player1)
[if playNum = 2]
setOwner(Player :: player2)
Continues for all players (aka, Math.random() returns a number b/w 0 and playerNum)

findVillages

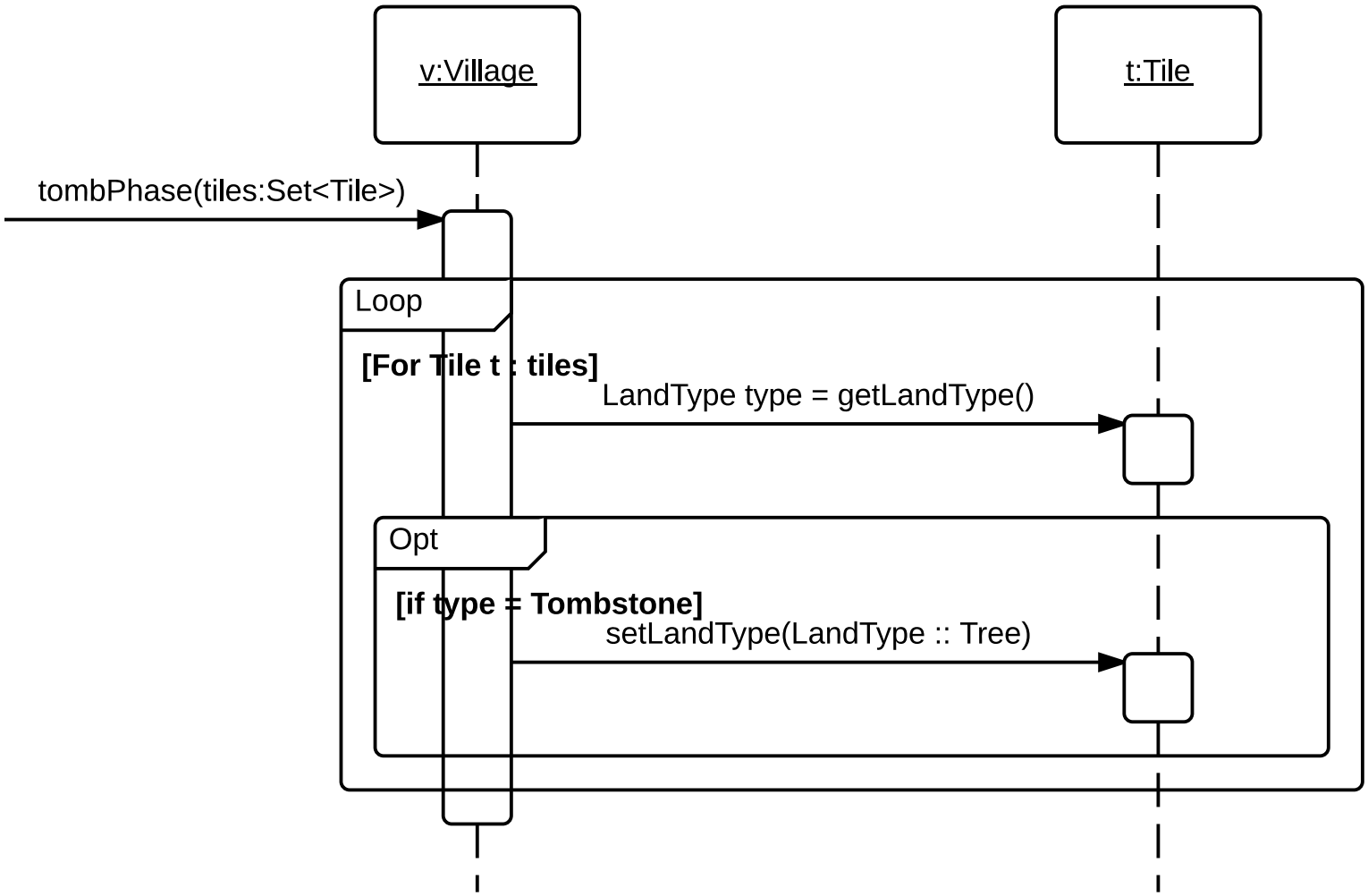



v:Village
t:Tile
strucTile:Tile
s:Structure
u:Unit
unitTile:Tile
delete
delete()
Set<Tile> tiles
= getTiles()
Loop
[for Tile t : tiles]
setOwner(null)
Tile now has no
owner (aka
neutral territory)
Tile strucTile =
getStrucTile()
Opt
[if strucTile != null]
removeStruct()
<delete>
Set<Unit> units
= getUnits()
Loop
[for Unit u : units]
Tile unitTile = getTile()
setLandType(LandType :: Tombstone)
removeUnit(u)
<delete>

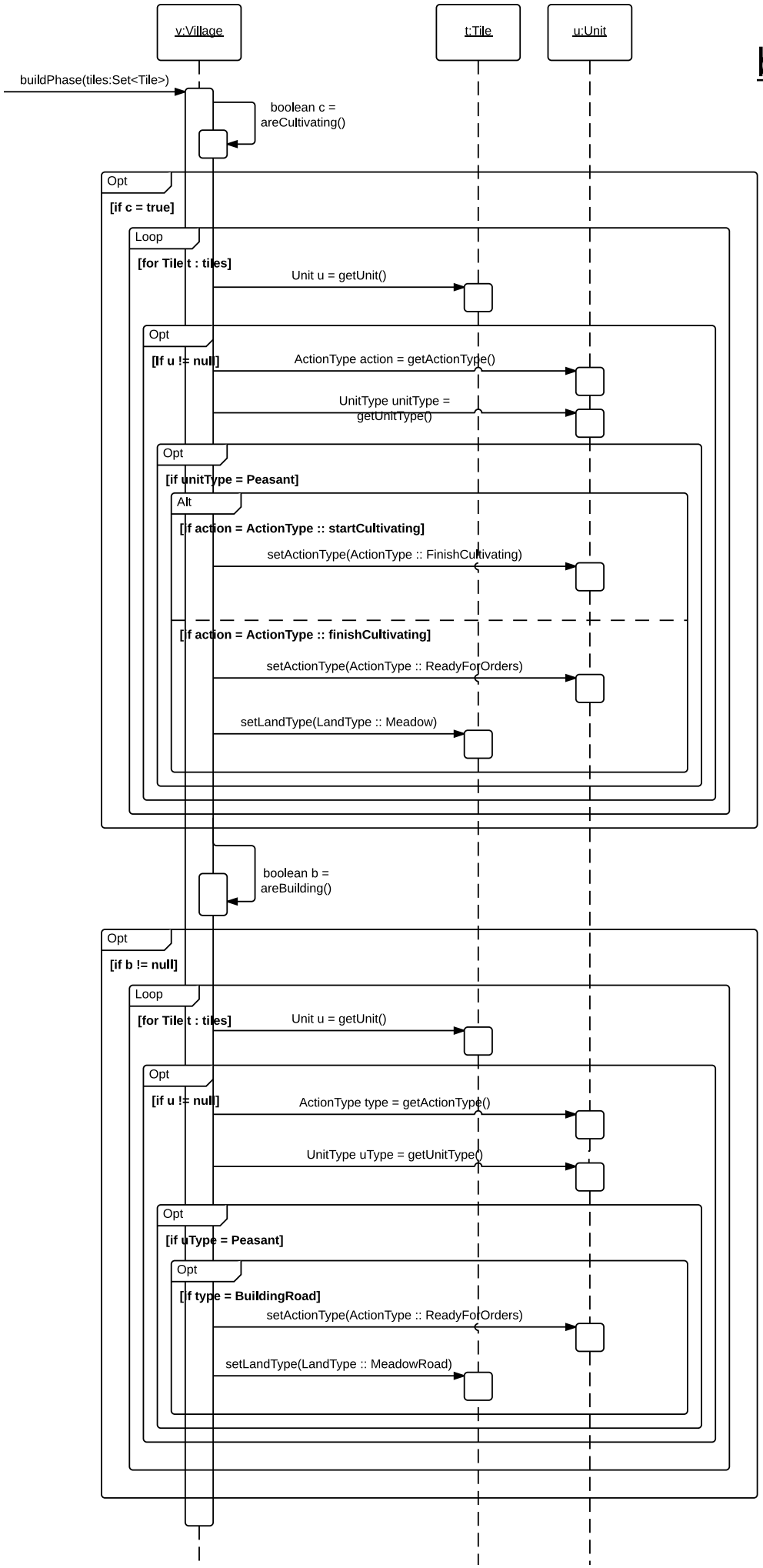
Begin Turn

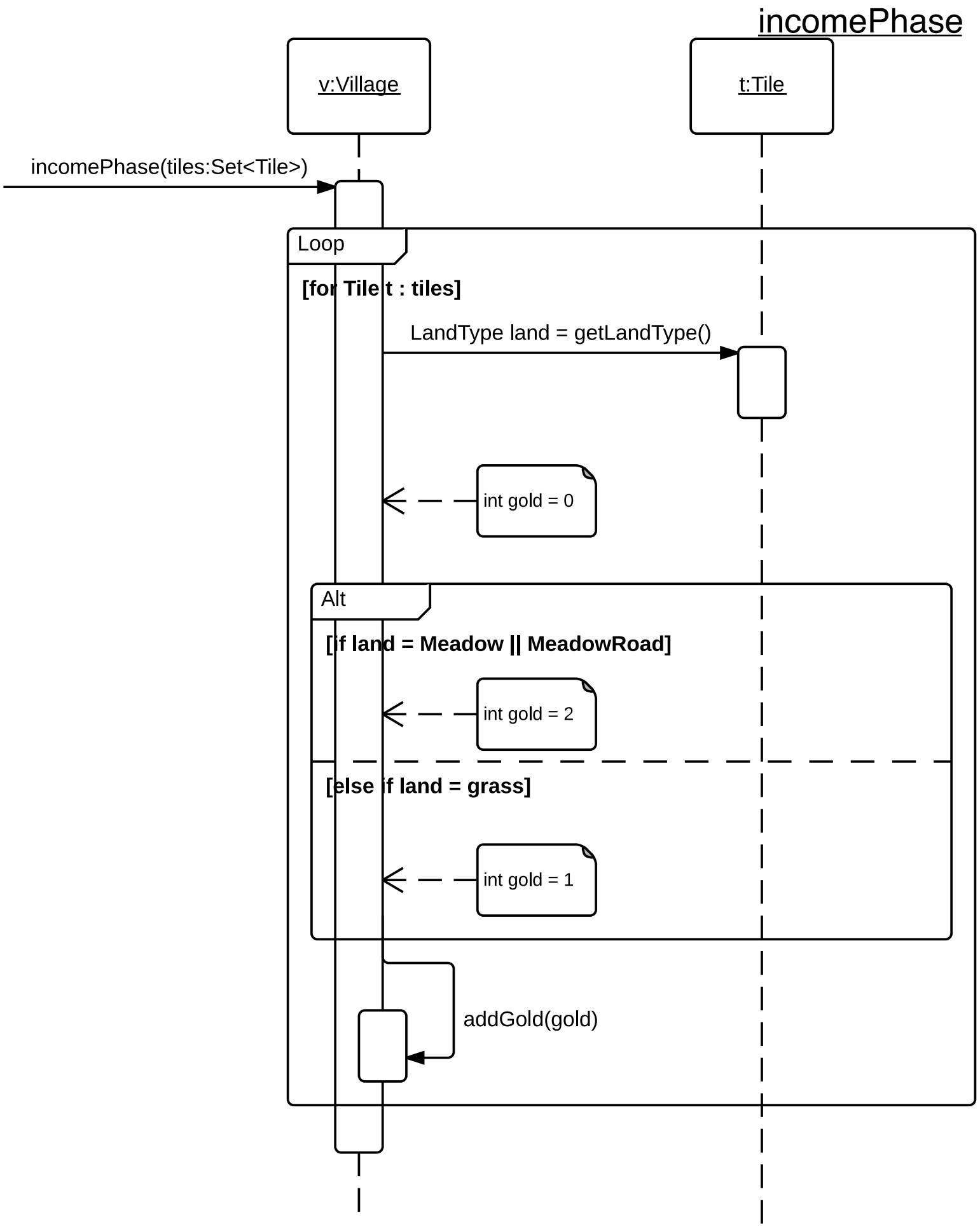

g:Game
p:Player
v:Village
beginTurn
(p: Player)
setActive()
Set<Village> villages = getVillages()
Loop
[for Village v: villages]
Set<Tile> tiles = getTiles()
refresh()
tombPhase(tiles)
buildPhase(tiles)
incomePhase(tiles)
paymentPhase(tiles)
takeTurn()

tombPhase

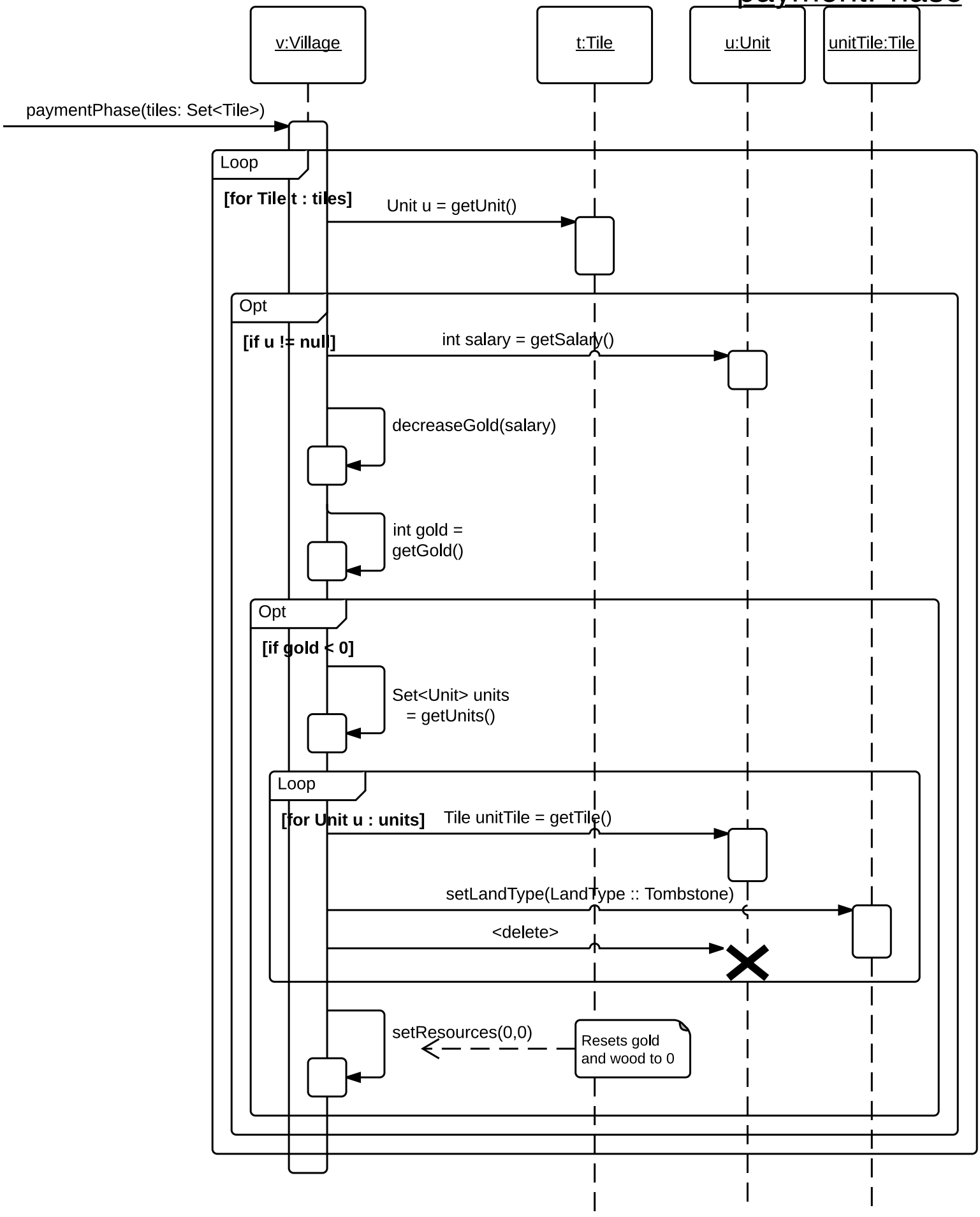

v:Village
t:Tile
tombPhase(tiles:Set<Tile>)
Loop
[For Tile t : tiles]
LandType type = getLandType()
Opt
[if type = Tombstone]
setLandType(LandType :: Tree)

buildPhase

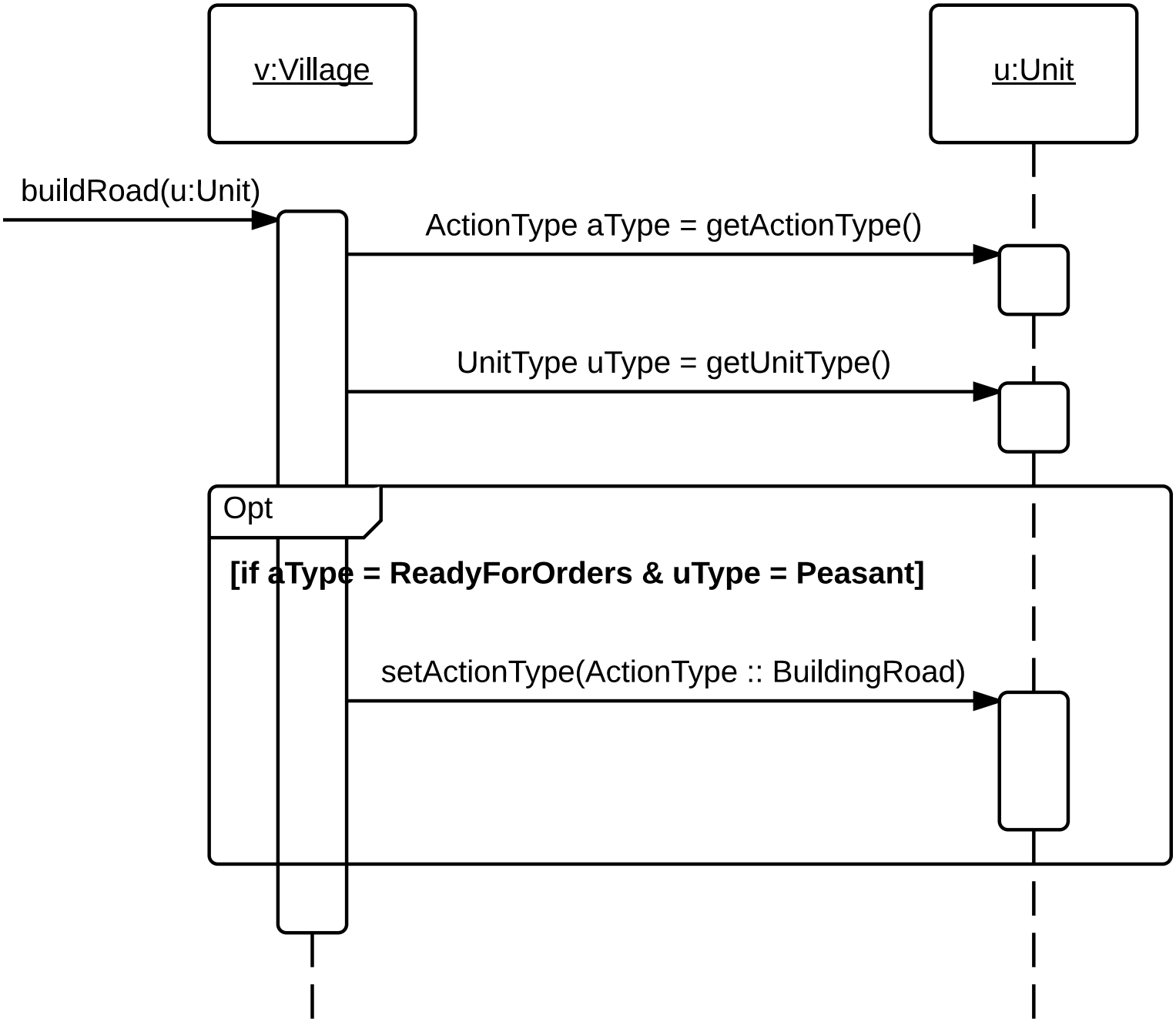

v:Village
t:Tile
u:Unit
buildPhase(tiles:Set<Tile>)
boolean c = areCultivating()
Opt
[if c = true]
Loop
[for Tile t : tiles]
Unit u = getUnit()
Opt
[If u != null]
ActionType action = getActionType()
UnitType unitType = getUnitType()
Opt
[if unitType = Peasant]
Alt
[if action = ActionType :: startCultivating]
setActionType(ActionType :: FinishCultivating)
[if action = ActionType :: finishCultivating]
setActionType(ActionType :: ReadyForOrders)
setLandType(LandType :: Meadow)
boolean b = areBuilding()
Opt
[if b != null]
Loop
[for Tile t : tiles]
Unit u = getUnit()
Opt
[if u != null]
ActionType type = getActionType()
UnitType uType = getUnitType()
Opt
[if uType = Peasant]
Opt
[if type = BuildingRoad]
setActionType(ActionType :: ReadyForOrders)
setLandType(LandType :: MeadowRoad)


incomePhase
v:Village
t:Tile
incomePhase(tiles:Set<Tile>)
Loop
[for Tile t : tiles]
LandType land = getLandType()
int gold = 0
Alt
[if land = Meadow || MeadowRoad]
int gold = 2
[else if land = grass]
int gold = 1
addGold(gold)

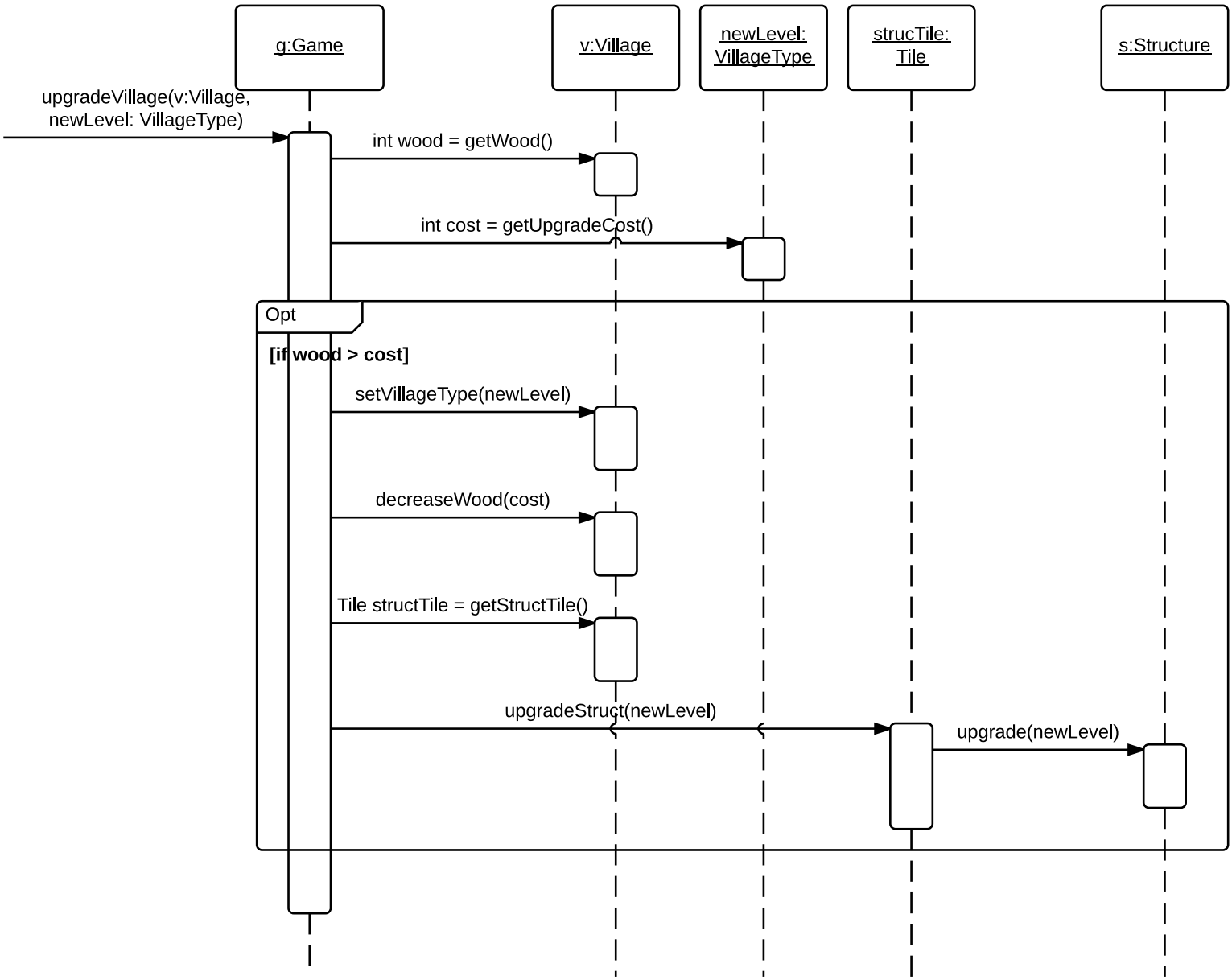
paymentPhase



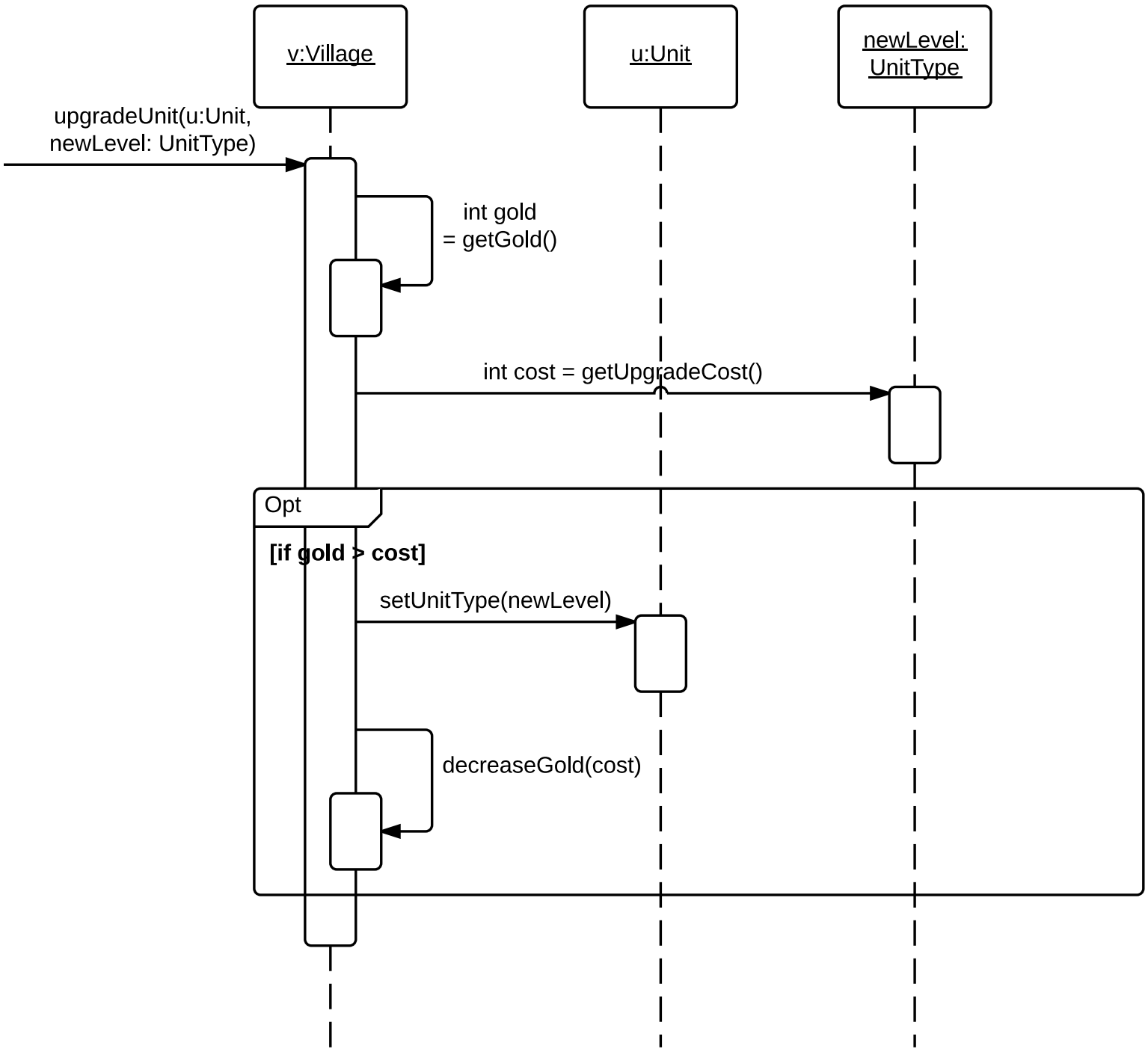
Build Road


v:Village
u:Unit
buildRoad(u:Unit)
ActionType aType = getActionType()
UnitType uType = getUnitType()
Opt
[if aType = ReadyForOrders & uType = Peasant]
setActionType(ActionType :: BuildingRoad)

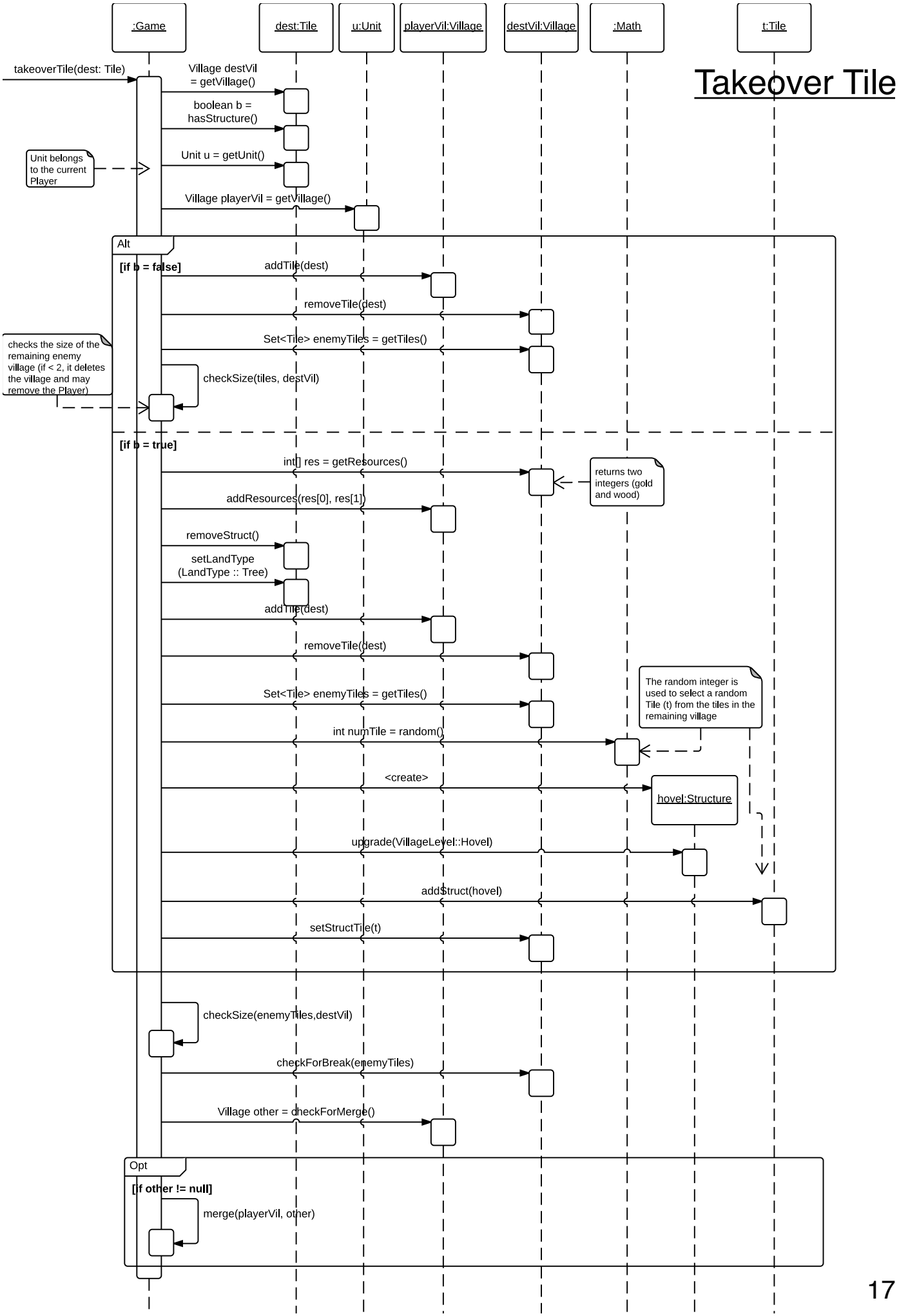
Upgrade Village

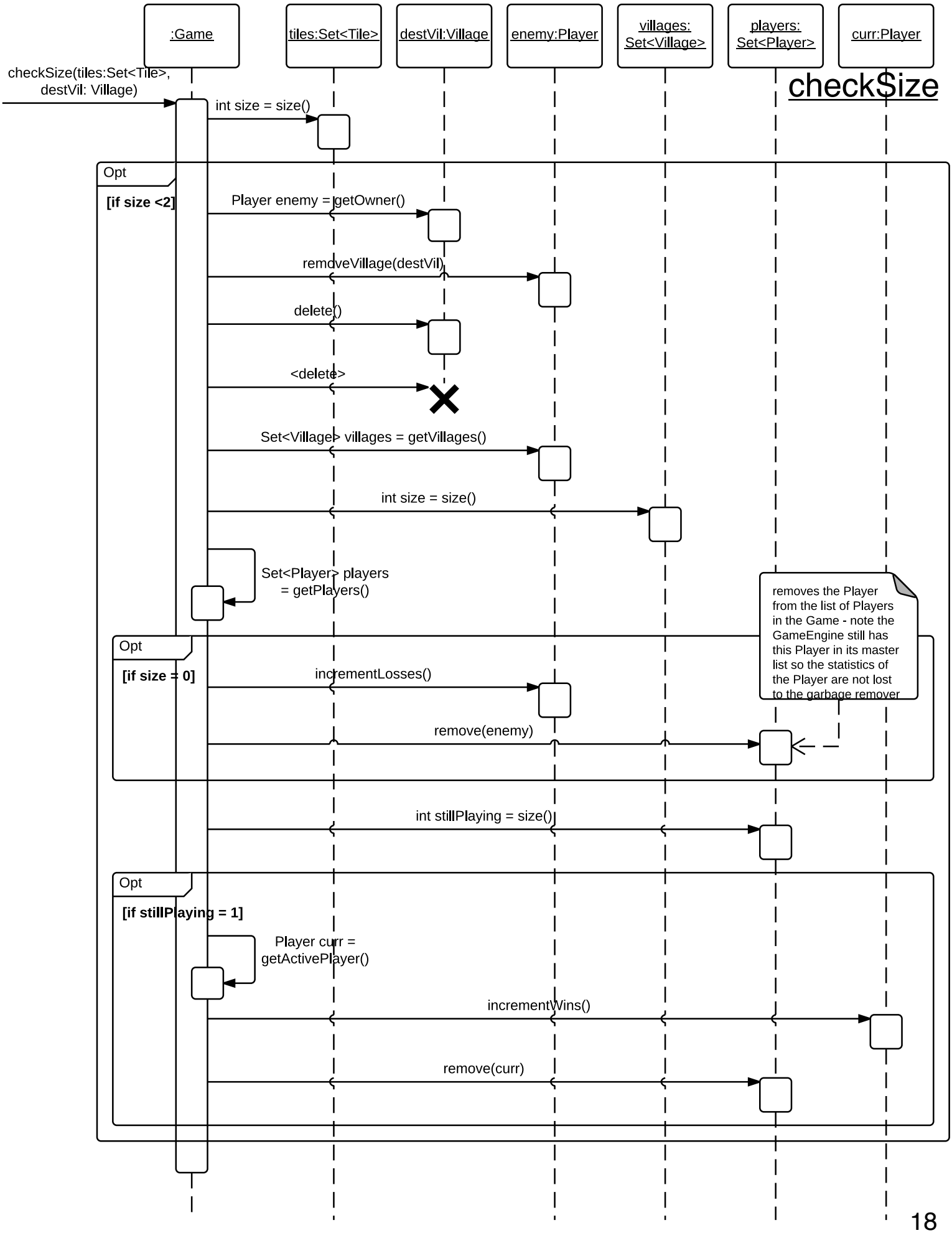


Upgrade Unit

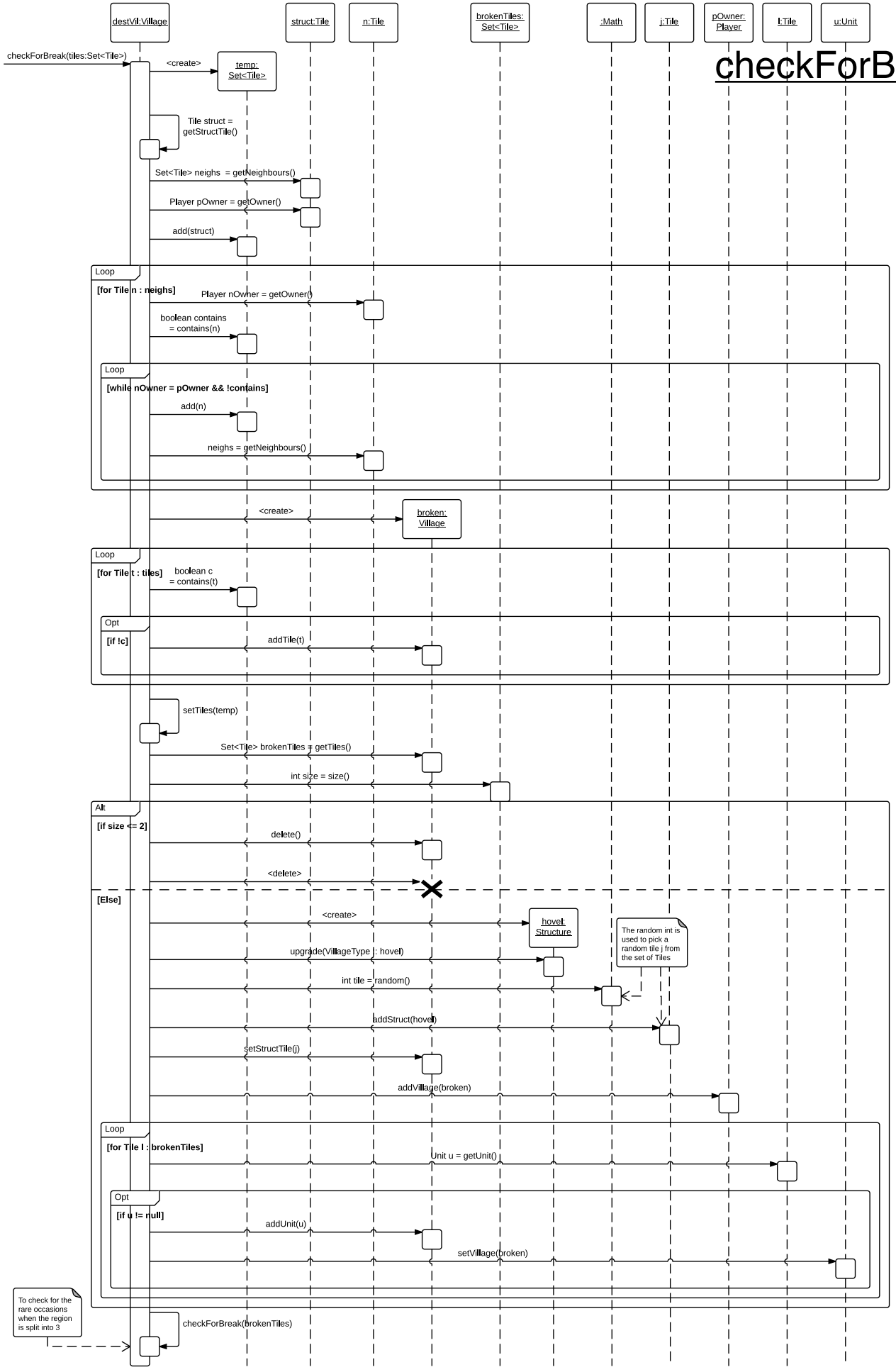


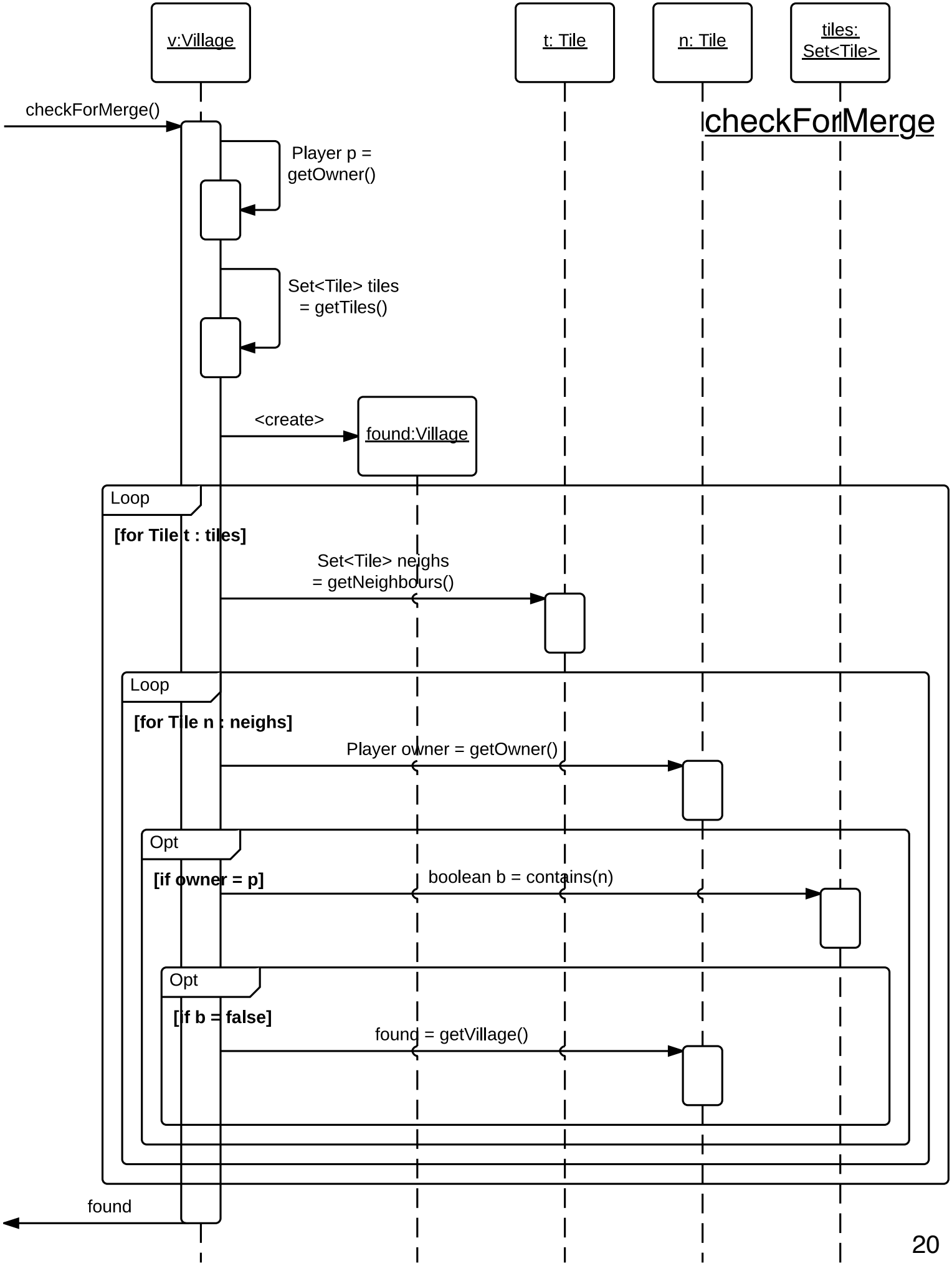
Takeover Tile



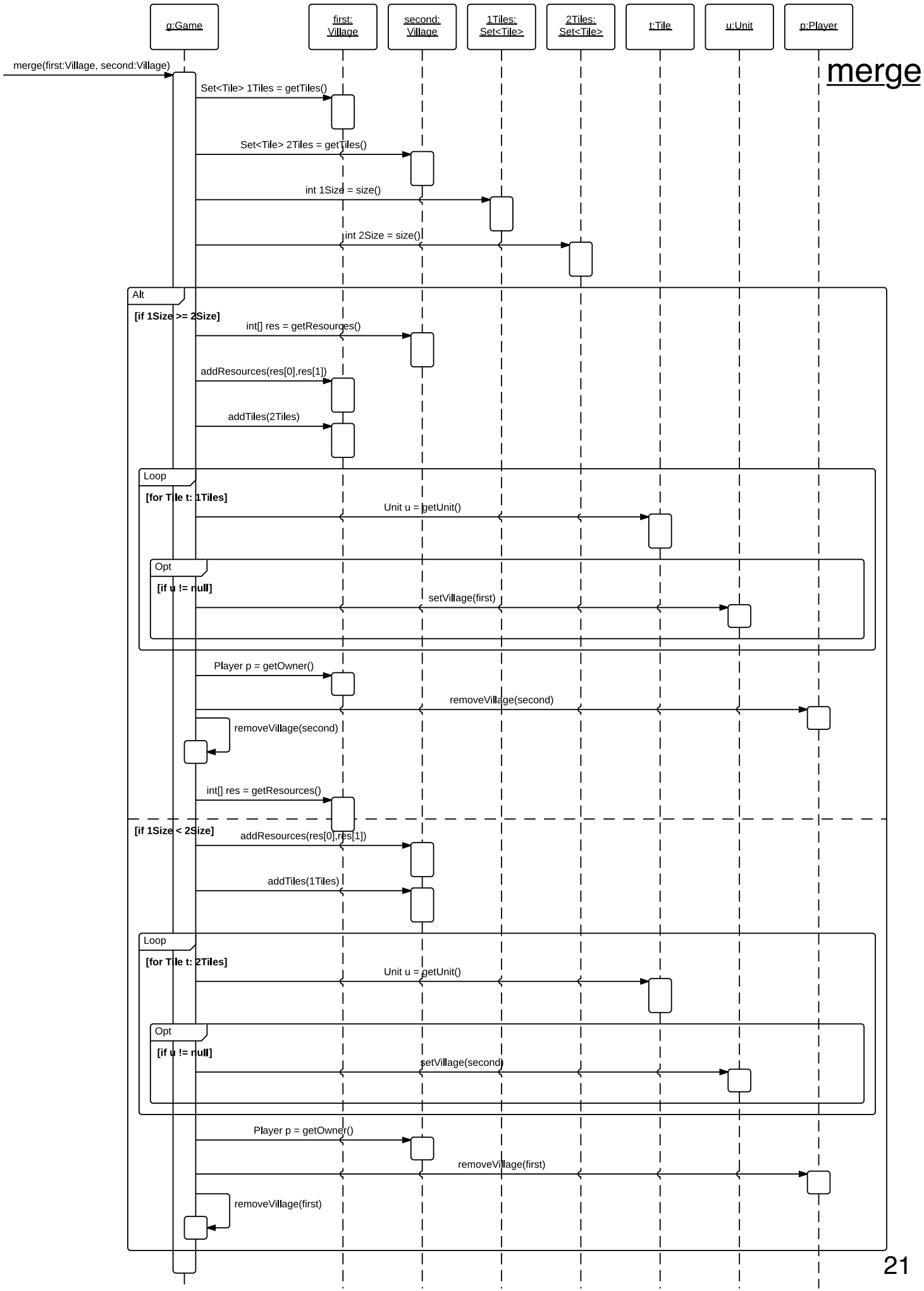

:Game
tiles:Set<Tile>
destVil:Village
enemy:Player
villages: Set<Village>
players: Set<Player>
curr:Player
checkSize
checkSize(tiles:Set<Tile>, destVil: Village)
int size = size()
Opt
[if size <2]
Player enemy = getOwner()
removeVillage(destVil)
delete()
<delete>
Set<Village> villages = getVillages()
int size = size()
Set<Player> players = getPlayers()
removes the Player from the list of Players in the Game - note the GameEngine still has this Player in its master list so the statistics of the Player are not lost to the garbage remover
Opt
[if size = 0]
incrementLosses()
remove(enemy)
int stillPlaying = size()
Opt
[if stillPlaying = 1]
Player curr = getActivePlayer()
incrementWins()
remove(curr)

checkForBreak

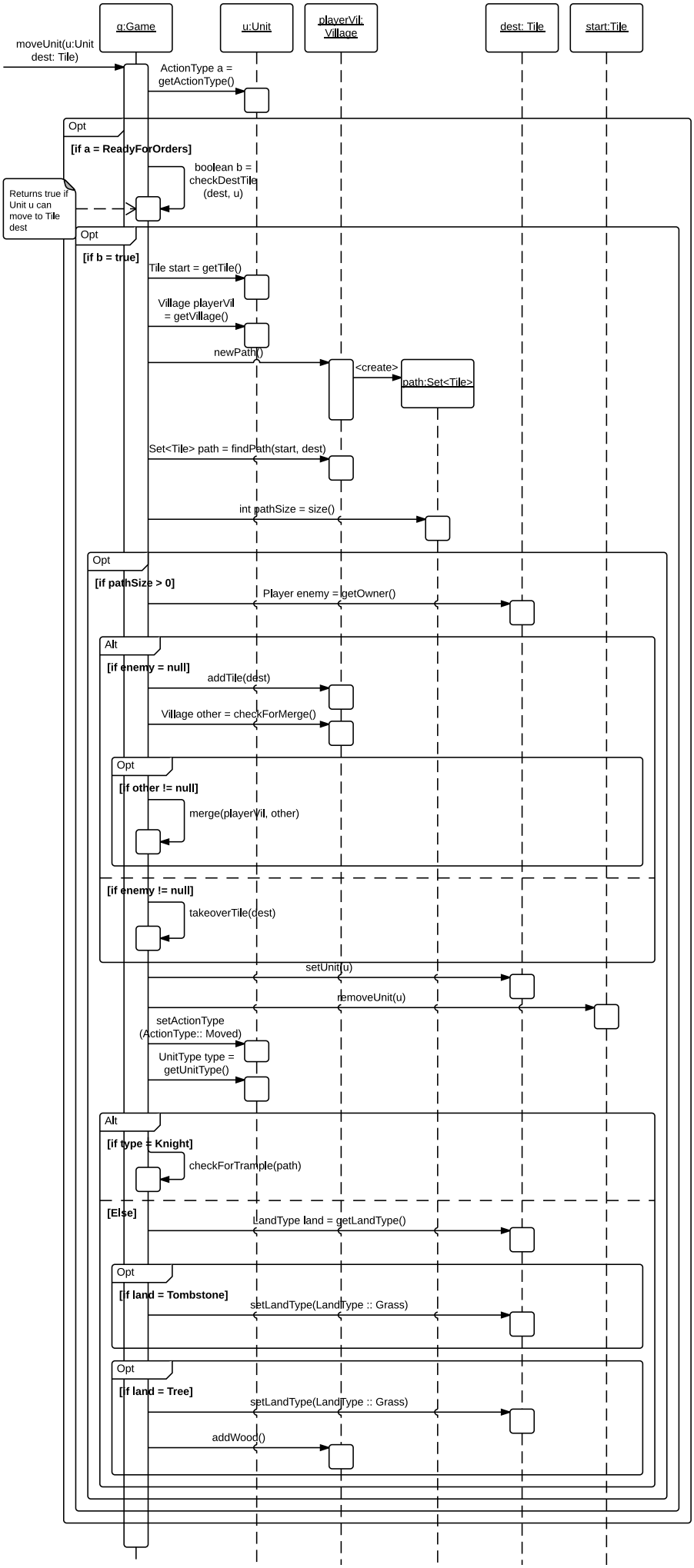



v:Village
t: Tile
n: Tile
tiles: Set<Tile>
checkForMerge()
checkForMerge
Player p = getOwner()
Set<Tile> tiles = getTiles()
<create>
found:Village
Loop
[for Tile t : tiles]
Set<Tile> neighs = getNeighbours()
Loop
[for Tile n : neighs]
Player owner = getOwner()
Opt
[if owner = p]
boolean b = contains(n)
Opt
[if b = false]
found = getVillage()
found

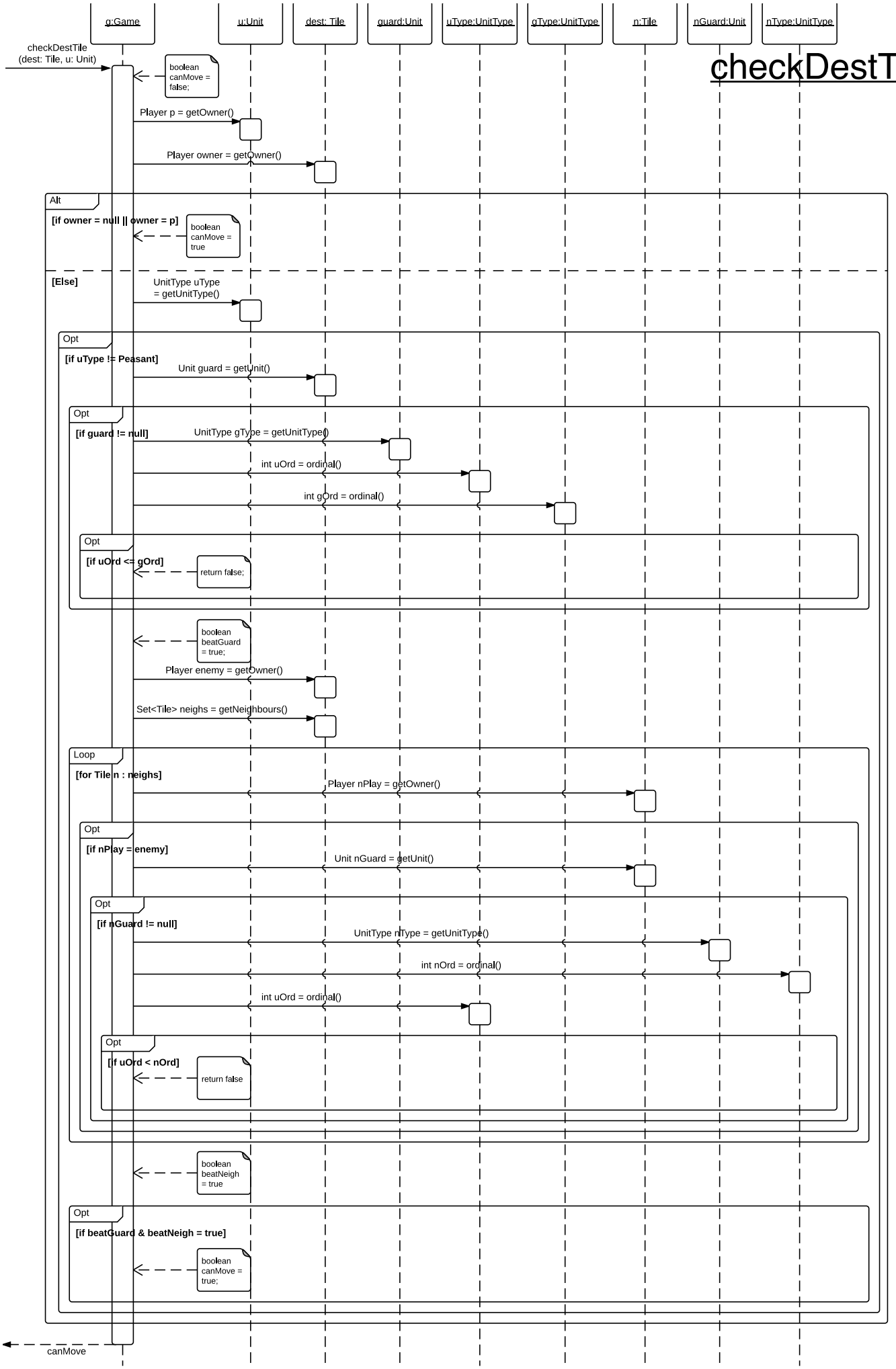
merge


g:Game
first: Village
second: Village
1Tiles: Set<Tile>
2Tiles: Set<Tile>
t:Tile
u:Unit
p:Player
merge(first:Village, second:Village)
Set<Tile> 1Tiles = getTiles()
Set<Tile> 2Tiles = getTiles()
int 1Size = size()
int 2Size = size()
Alt
[if 1Size >= 2Size]
int[] res = getResources()
addResources(res[0],res[1])
addTiles(2Tiles)
Loop
[for Tile t: 1Tiles]
Unit u = getUnit()
Opt
[if u != null]
setVillage(first)
Player p = getOwner()
removeVillage(second)
removeVillage(second)
int[] res = getResources()
[if 1Size < 2Size]
addResources(res[0],res[1])
addTiles(1Tiles)
Loop
[for Tile t: 2Tiles]
Unit u = getUnit()
Opt
[if u != null]
setVillage(second)
Player p = getOwner()
removeVillage(first)
removeVillage(first)

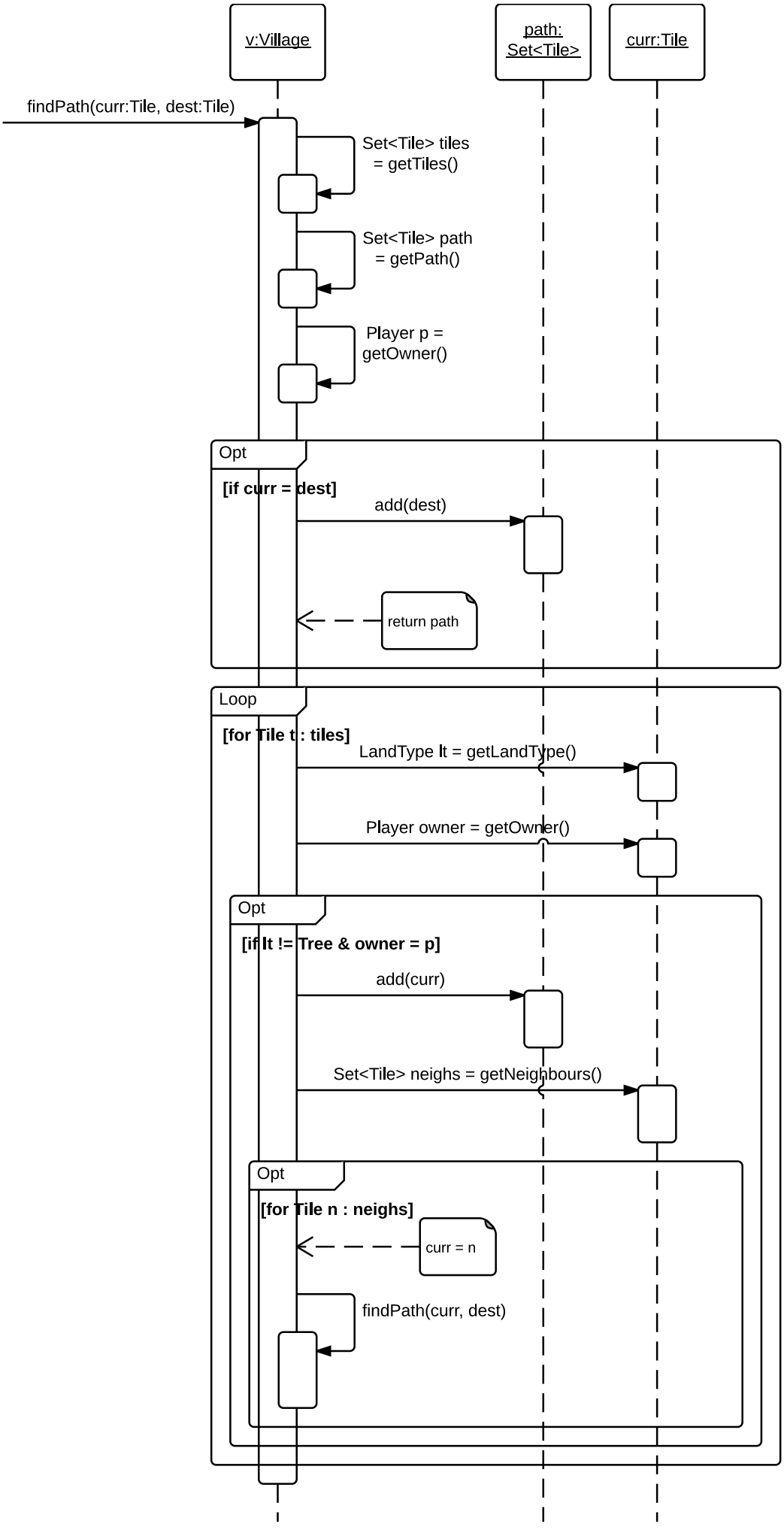
Move Unit


g:Game
u:Unit
playerVil: Village
dest: Tile
start:Tile
moveUnit(u:Unit dest: Tile)
ActionType a = getActionType()
Opt
[if a = ReadyForOrders]
boolean b = checkDestTile (dest, u)
Returns true if Unit u can move to Tile dest
Opt
[if b = true]
Tile start = getTile()
Village playerVil = getVillage()
newPath()
<create>
path:Set<Tile>
Set<Tile> path = findPath(start, dest)
int pathSize = size()
Opt
[if pathSize > 0]
Player enemy = getOwner()
Alt
[if enemy = null]
addTile(dest)
Village other = checkForMerge()
Opt
[if other != null]
merge(playerVil, other)
[if enemy != null]
takeoverTile(dest)
setUnit(u)
removeUnit(u)
setActionType (ActionType:: Moved)
UnitType type = getUnitType()
Alt
[if type = Knight]
checkForTrample(path)
[Else]
LandType land = getLandType()
Opt
[if land = Tombstone]
setLandType(LandType :: Grass)
Opt
[if land = Tree]
setLandType(LandType :: Grass)
addWood()

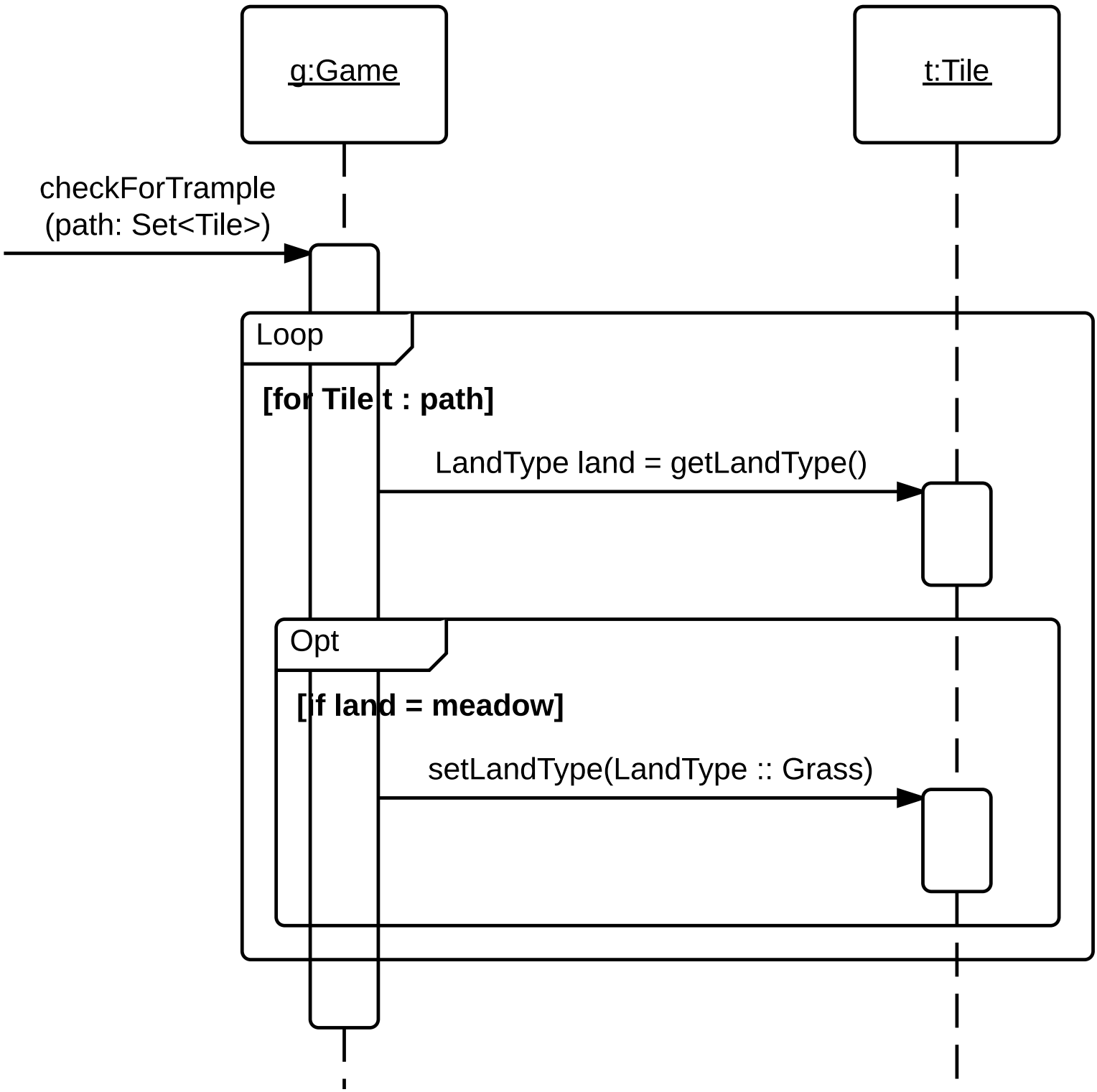
checkDestTile



findPath


v:Village
path:
Set<Tile>
curr:Tile
findPath(curr:Tile, dest:Tile)
Set<Tile> tiles
= getTiles()
Set<Tile> path
= getPath()
Player p =
getOwner()
Opt
[if curr = dest]
add(dest)
return path
Loop
[for Tile t : tiles]
LandType lt = getLandType()
Player owner = getOwner()
Opt
[if lt != Tree & owner = p]
add(curr)
Set<Tile> neighs = getNeighbours()
Opt
[for Tile n : neighs]
curr = n
findPath(curr, dest)

checkForTrample



Class Diagram