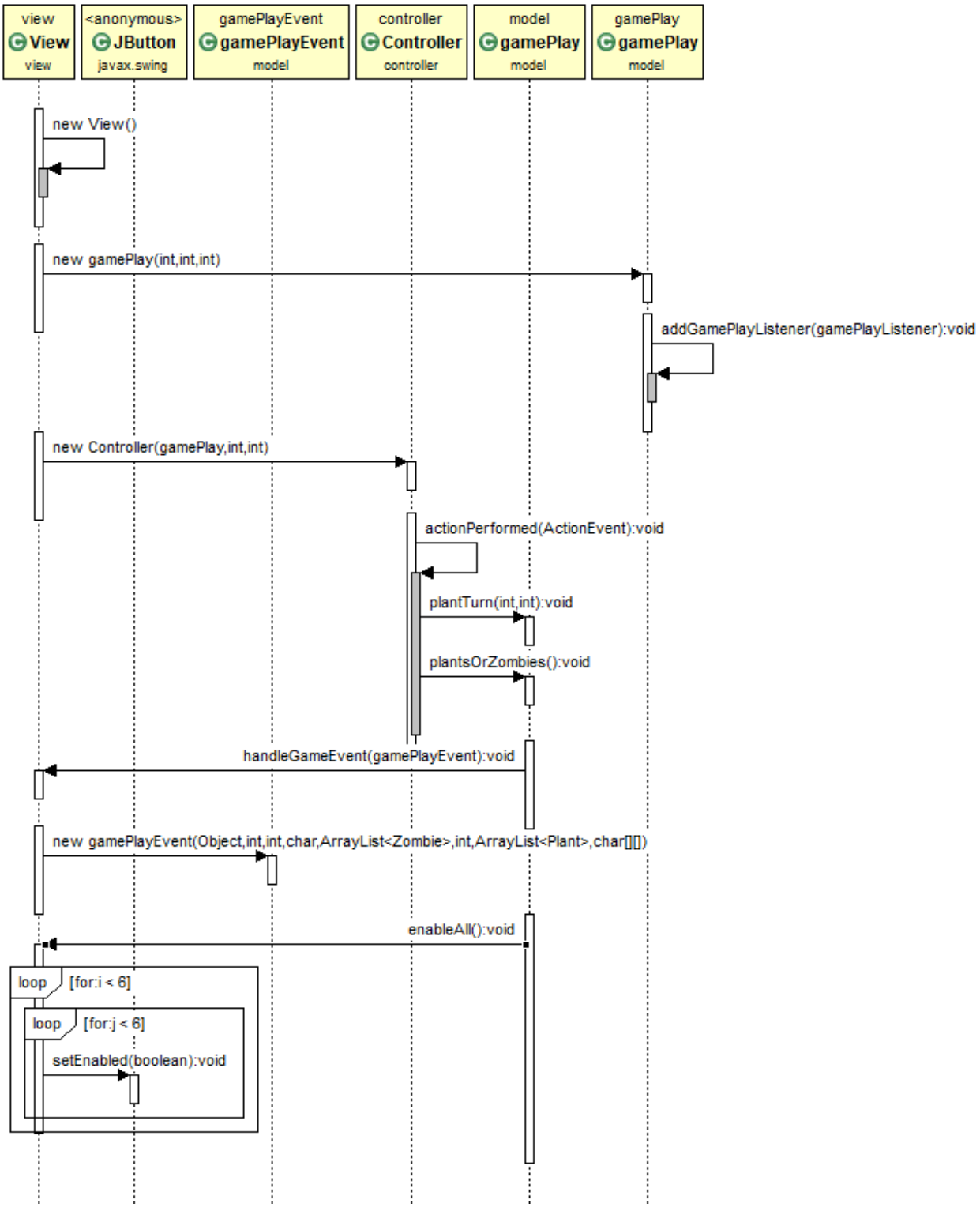


Diagram 1: A view is created when the user starts playing the game, creating a gamePlay (model) and controller which implements actionListener. The sequence of actions performed are shown in diagrams below. At the start, all buttons are enabled.

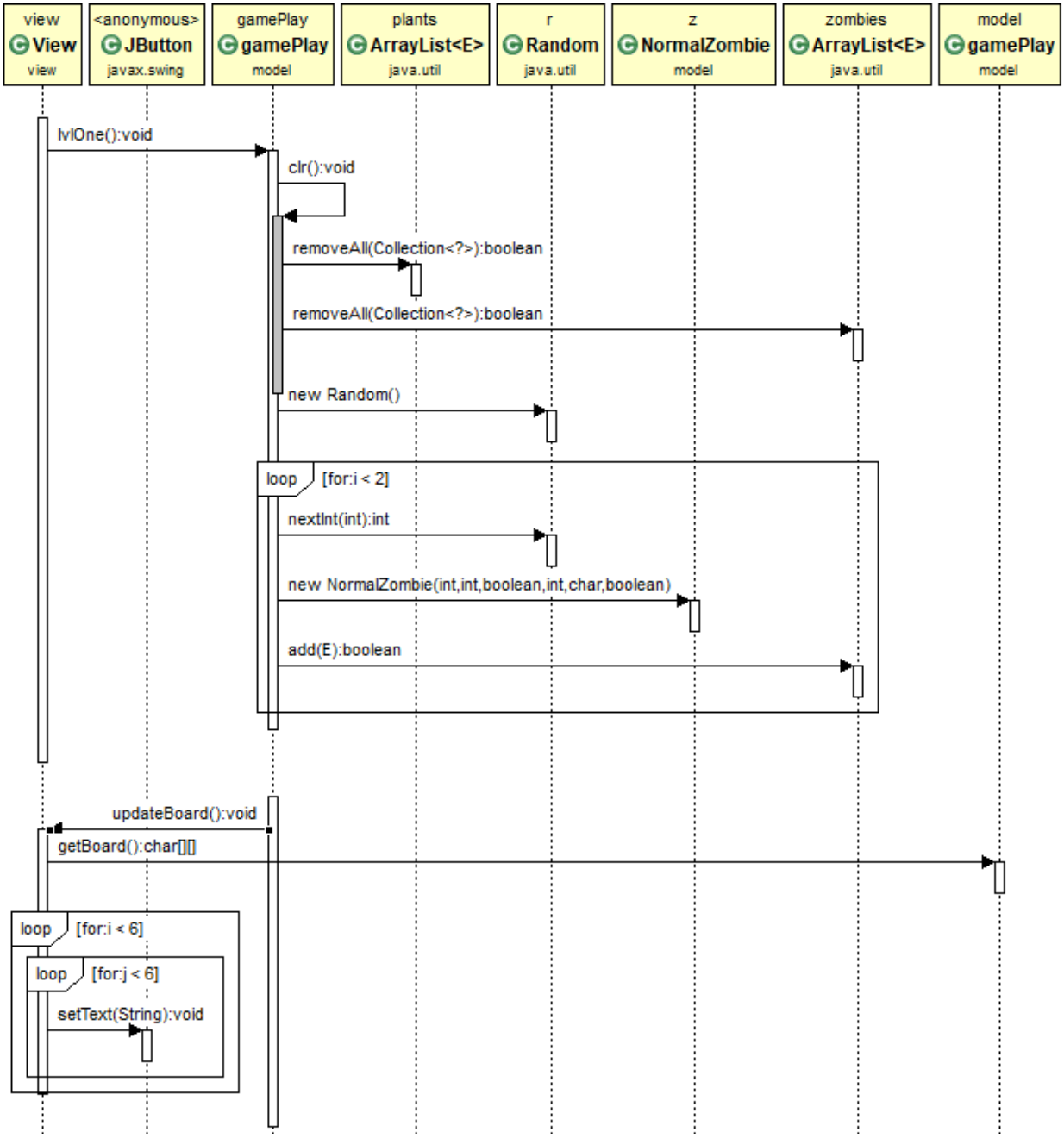


Diagram 2: A continuation from diagram 1 where the view interacts with the model in setting up level 1. The board is cleared (clr is called) and only normal zombies are present in this level. The board is then updated.

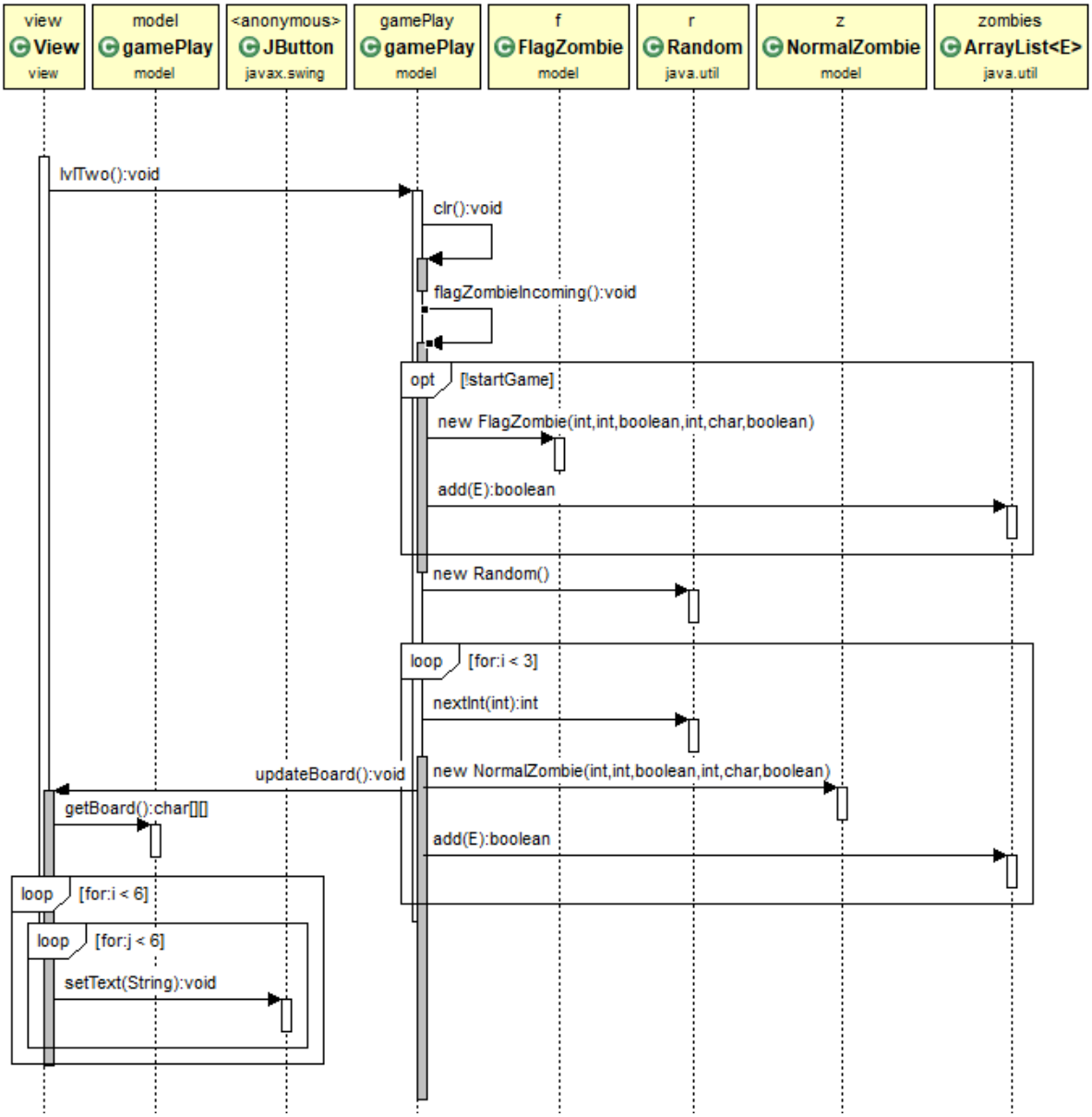


Diagram 3: A continuation from diagram 1 where the view interacts with the model in setting up level 2. The board is cleared (clr is called) and now flag zombies (flagZombieIncoming is invoked) as well as normal zombies are present in this level. The board is then updated.

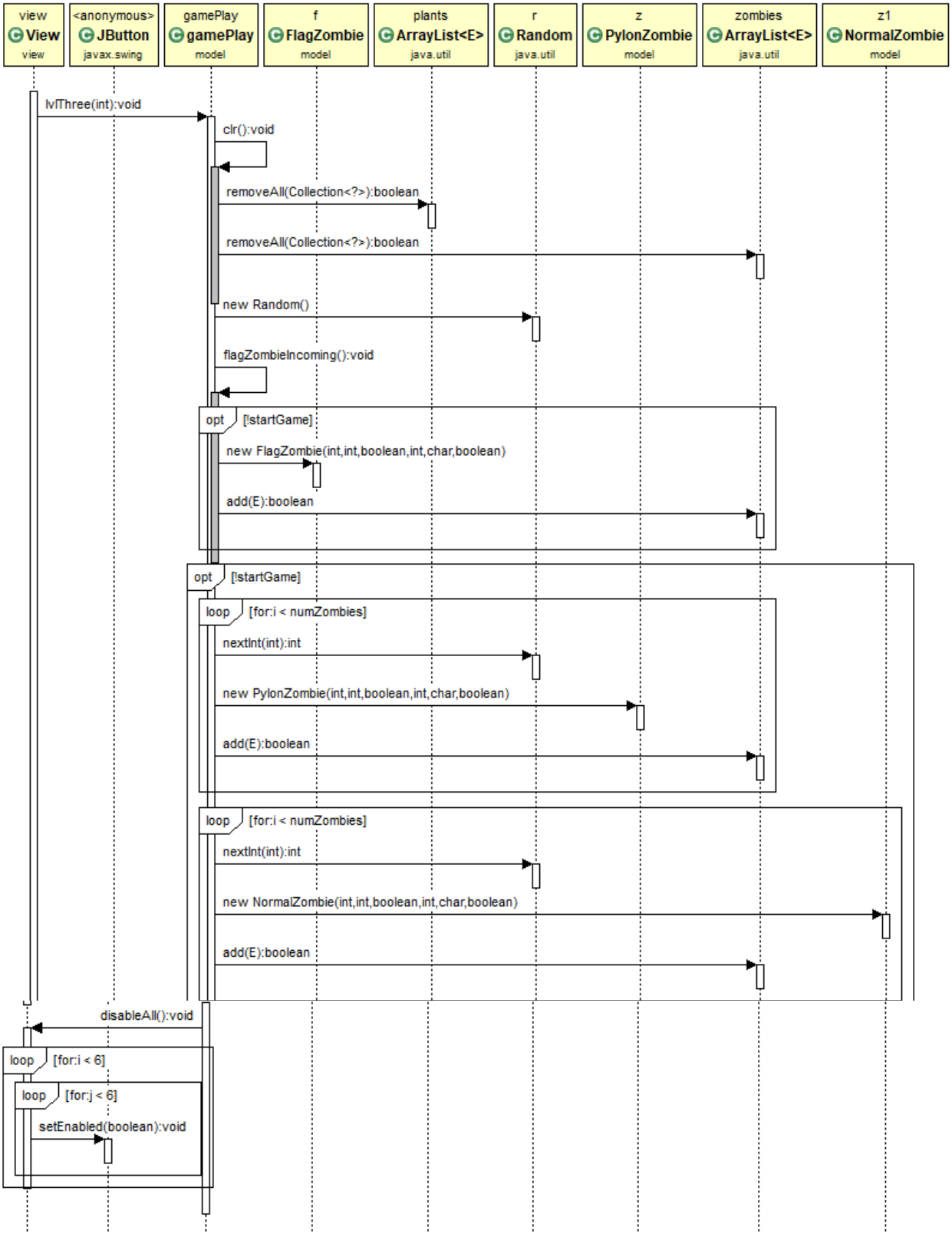
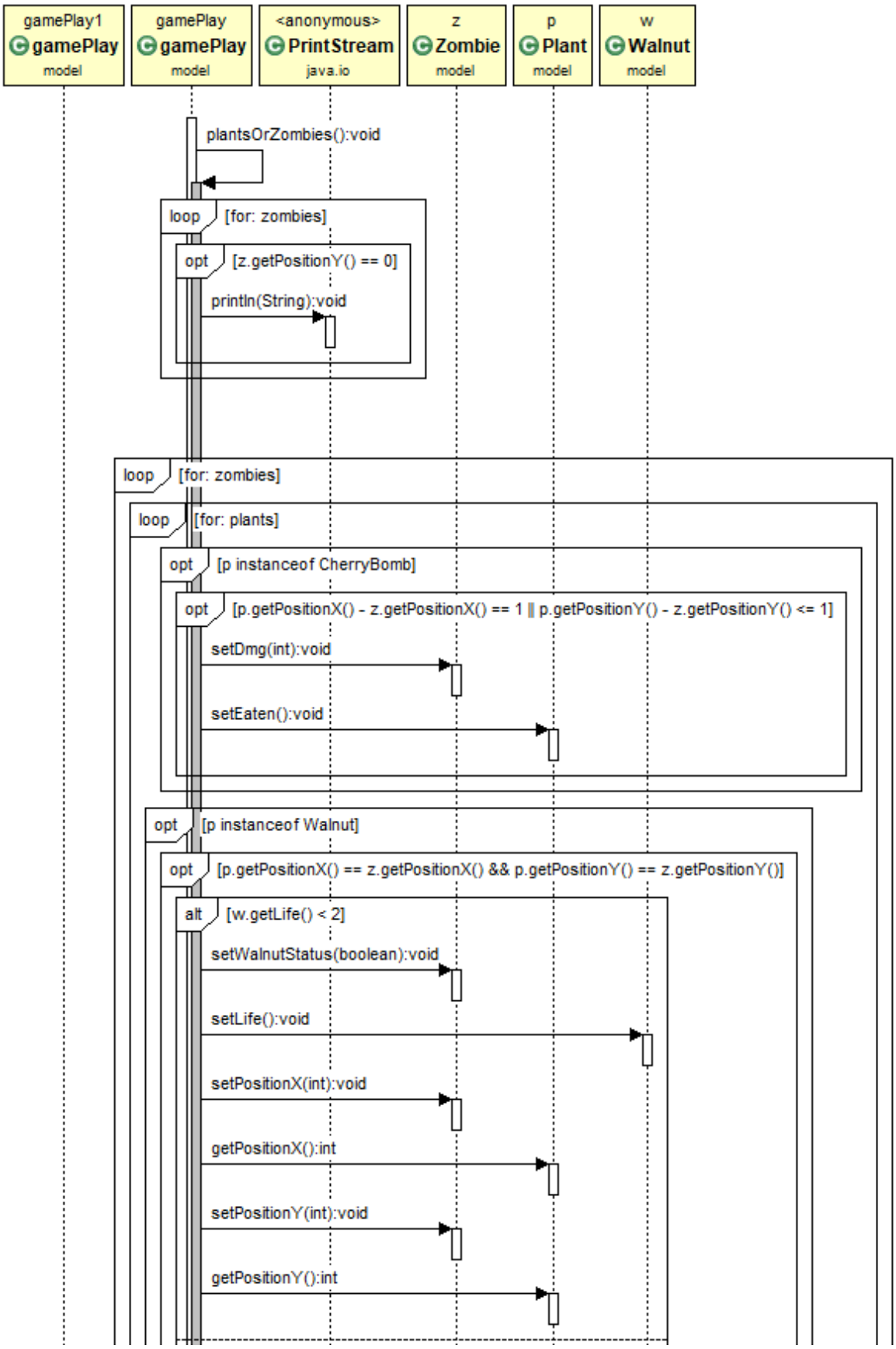


Diagram 4: A continuation from diagram 1 where the view interacts with the model in setting up level 3. The board is cleared (clr is called) and now pylon zombies as well as normal and flag zombies (flagZombieIncoming is invoked) are present in this level. The board is then updated.


gamePlay1
gamePlay
model
gamePlay
gamePlay
model
<anonymous>
PrintStream
java.io
z
Zombie
model
p
Plant
model
w
Walnut
model
plantsOrZombies():void
loop
[for: zombies]
opt
[z.getPositionY() == 0]
println(String):void
loop
[for: zombies]
loop
[for: plants]
opt
[p instanceof CherryBomb]
opt
[p.getPositionX() - z.getPositionX() == 1 || p.getPositionY() - z.getPositionY() <= 1]
setDmg(int):void
setEaten():void
opt
[p instanceof Walnut]
opt
[p.getPositionX() == z.getPositionX() && p.getPositionY() == z.getPositionY()]
alt
[w.getLife() < 2]
setWalnutStatus(boolean):void
setLife():void
setPositionX(int):void
getPositionX():int
setPositionY(int):void
getPositionY():int

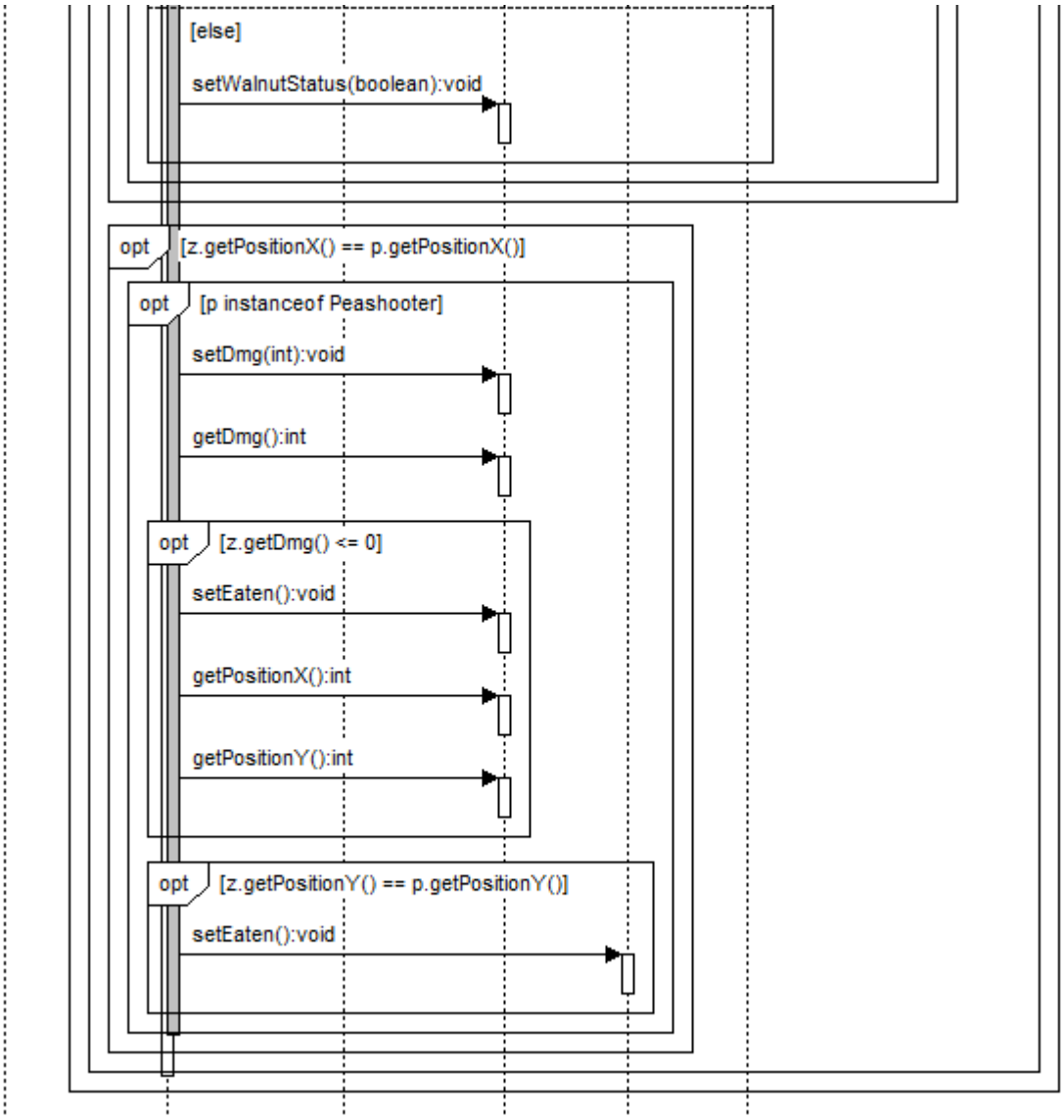


Diagram 5: This sequence diagram shows when plantsOrZombies method is called with the condition for each edge case. The edge cases are checking the positions of zombies in the game, the positions of plants in the game, the type of plant interacting and the level of damage points a zombie has.

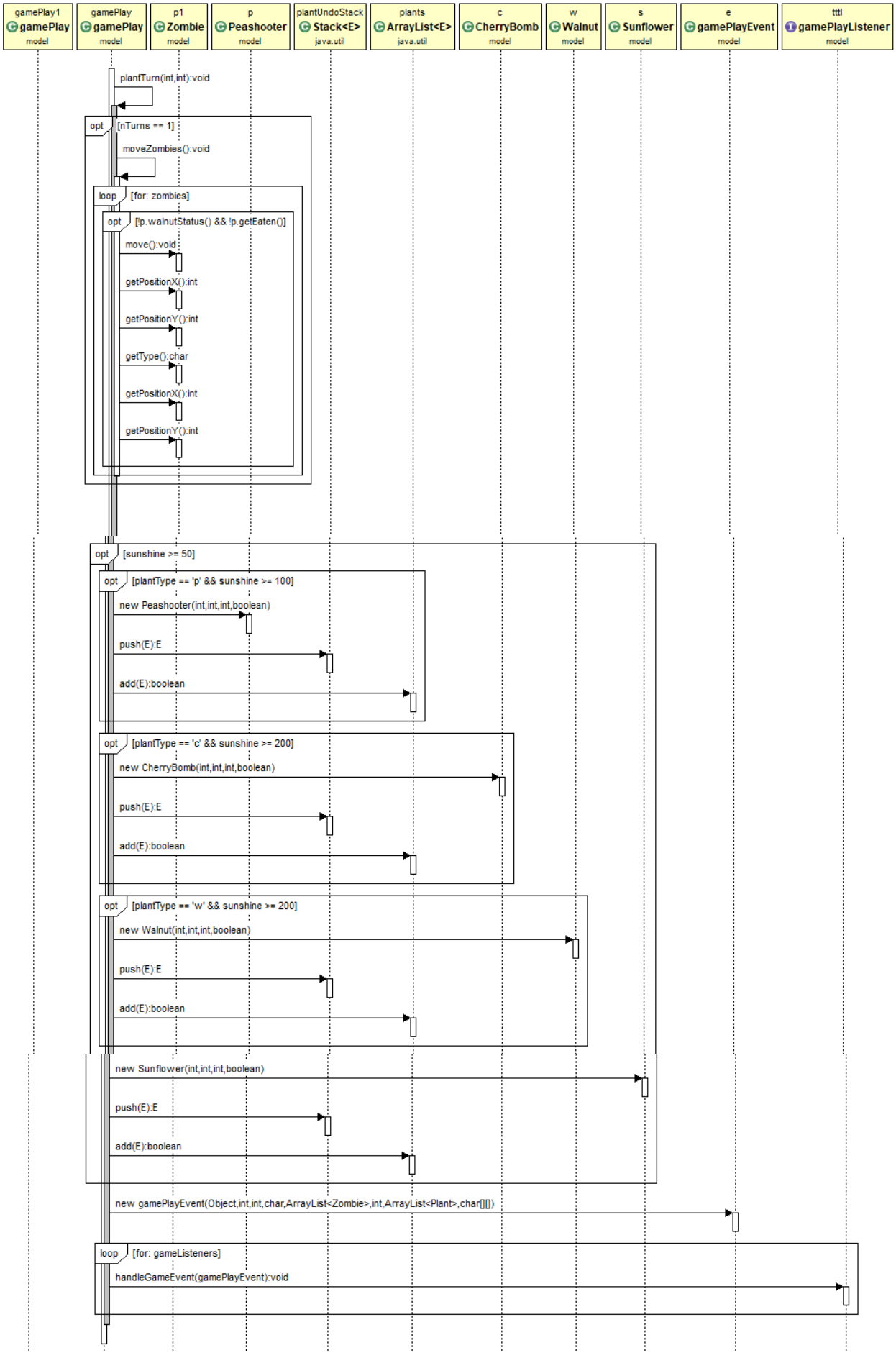

gamePlay1
gamePlay
model
gamePlay
gamePlay
model
p1
Zombie
model
p
Peashooter
model
plantUndoStack
Stack<E>
java.util
plants
ArrayList<E>
java.util
c
CherryBomb
model
w
Walnut
model
s
Sunflower
model
e
gamePlayEvent
model
tttl
gamePlayListener
model
plantTurn(int,int):void
opt
[nTurns == 1]
moveZombies():void
loop
[for: zombies]
opt
[!p.walnutStatus() && !p.getEaten()]
move():void
getPositionX():int
getPositionY():int
getType():char
getPositionX():int
getPositionY():int
opt
[sunshine >= 50]
opt
[plantType == 'p' && sunshine >= 100]
new Peashooter(int,int,int,boolean)
push(E):E
add(E):boolean
opt
[plantType == 'c' && sunshine >= 200]
new CherryBomb(int,int,int,boolean)
push(E):E
add(E):boolean
opt
[plantType == 'w' && sunshine >= 200]
new Walnut(int,int,int,boolean)
push(E):E
add(E):boolean
new Sunflower(int,int,int,boolean)
push(E):E
add(E):boolean
new gamePlayEvent(Object,int,int,char,ArrayList<Zombie>,int,ArrayList<Plant>,char[][])
loop
[for: gameListeners]
handleGameEvent(gamePlayEvent):void

Diagram 6 (above): This sequence diagram shows when plantTurn method is called with the condition for each edge case. For each plant turn, the zombies move as well (moveZombie is called). The edge cases are checking the amount of sunshines in the bank and if the choses type of plant is affordable enough.

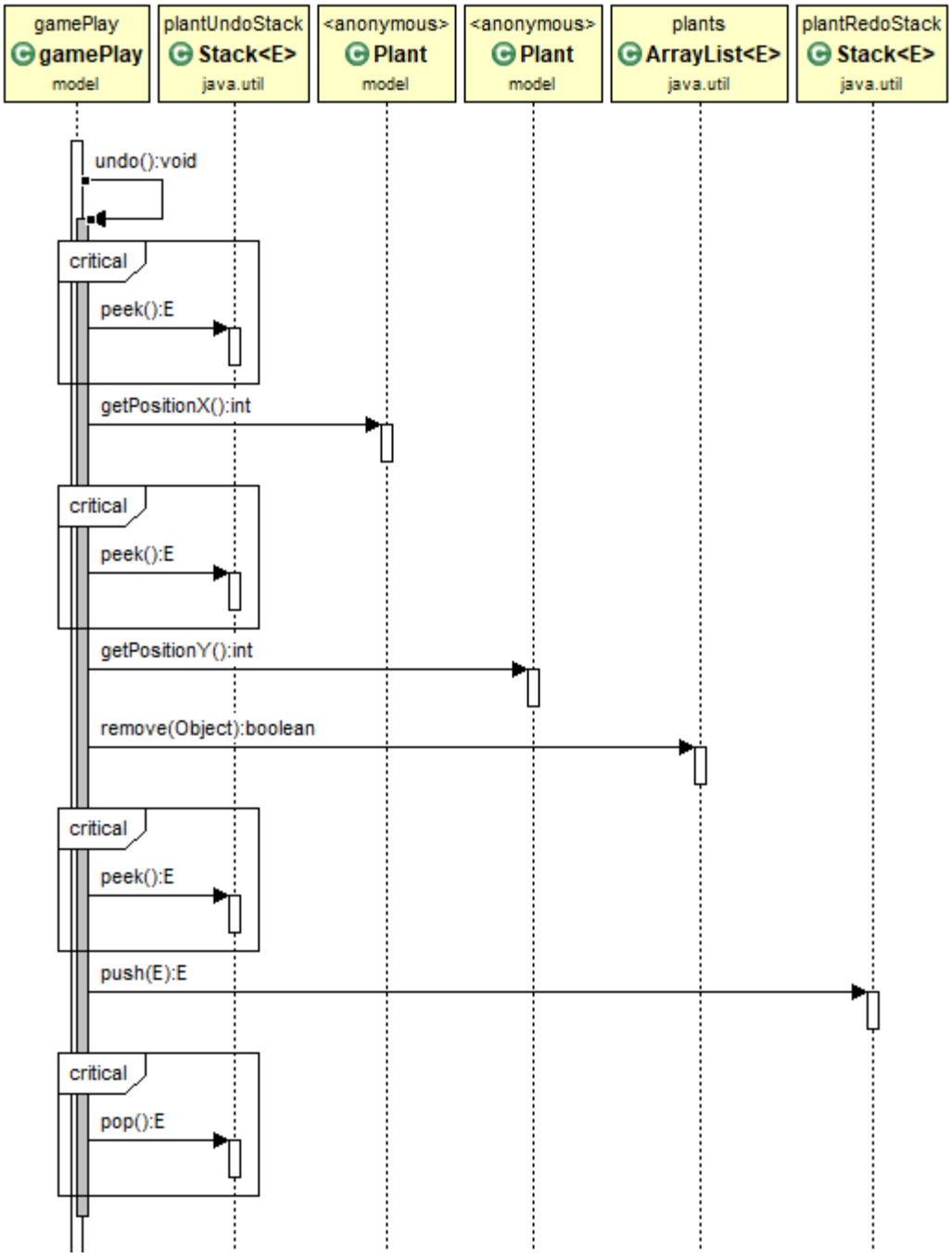


Diagram 7: This sequence diagram shows when undo method is called and the actions it performs, peeking into the stack, getting the top most plant/zombie and pushing back onto the board.

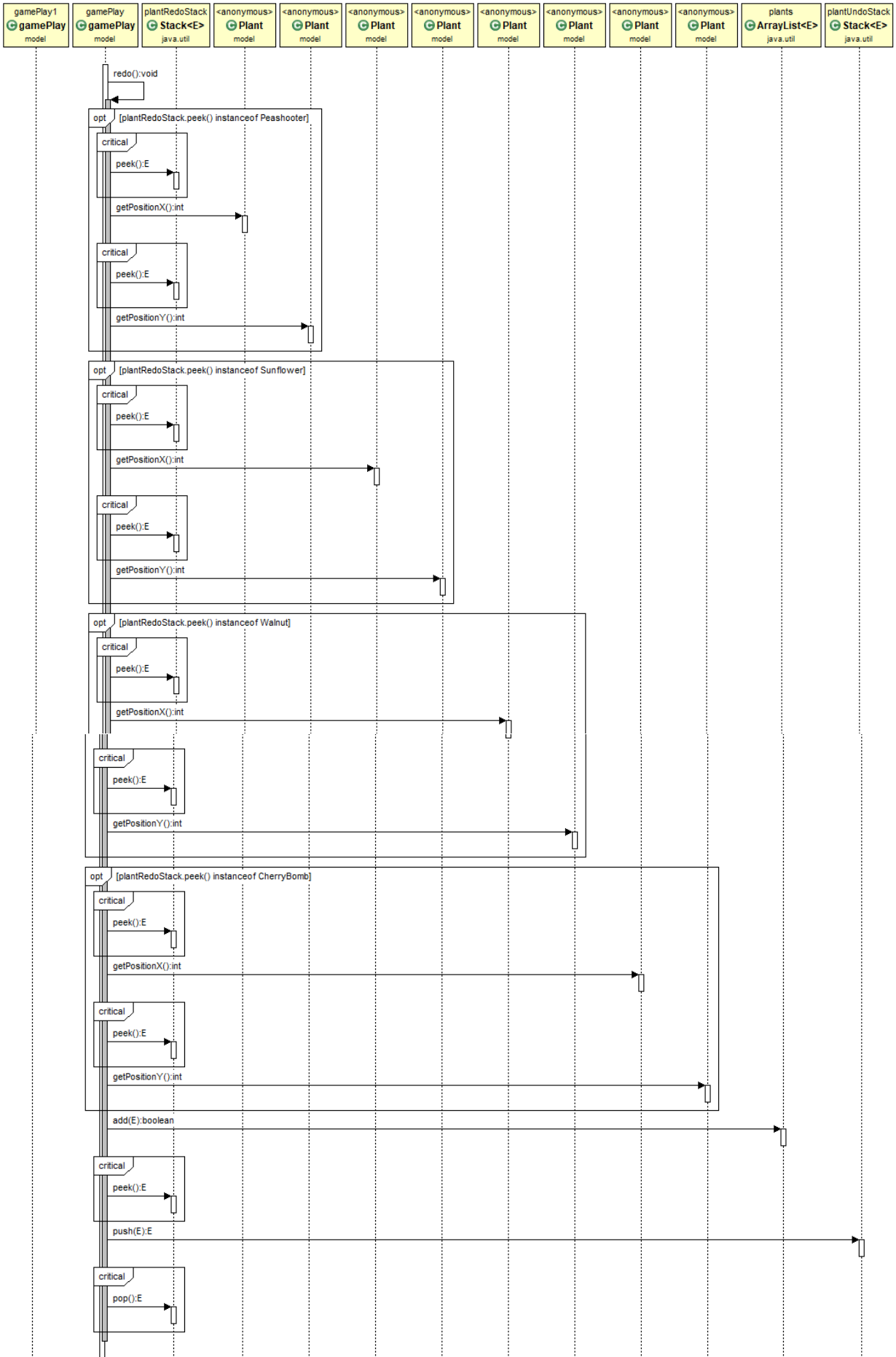

gamePlay1
gamePlay
model
gamePlay
gamePlay
model
plantRedoStack
Stack<E>
java.util
<anonymous>
Plant
model
plants
ArrayList<E>
java.util
plantUndoStack
Stack<E>
java.util
redo():void
opt
[plantRedoStack.peek() instanceof Peashooter]
critical
peek():E
getPositionX():int
getPositionY():int
[plantRedoStack.peek() instanceof Sunflower]
[plantRedoStack.peek() instanceof Walnut]
[plantRedoStack.peek() instanceof CherryBomb]
add(E):boolean
push(E):E
pop():E

Diagram 8 (above): This sequence diagram shows when redo method is called and the actions it performs according to each edge case by looking into the stack and comparing with the type of plant at the top. The position of the plant/zombie is gotten and pushed out onto the board.

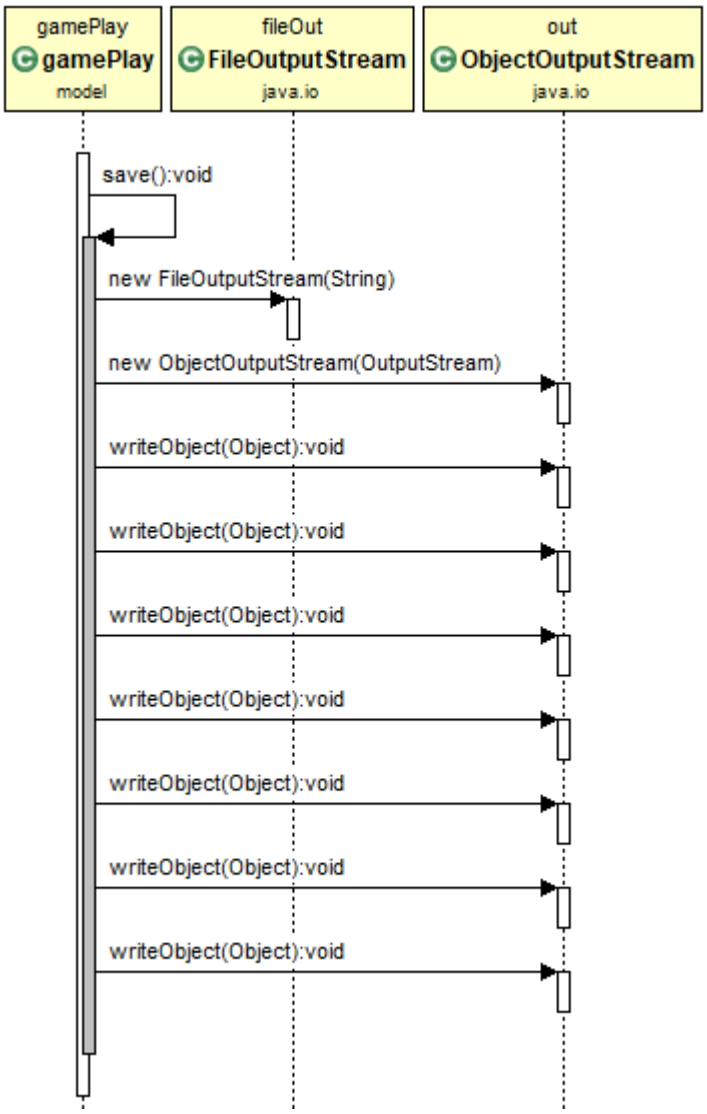


Diagram 9: This sequence diagram shows when save method is called by creating a new file output stream and writing to it.

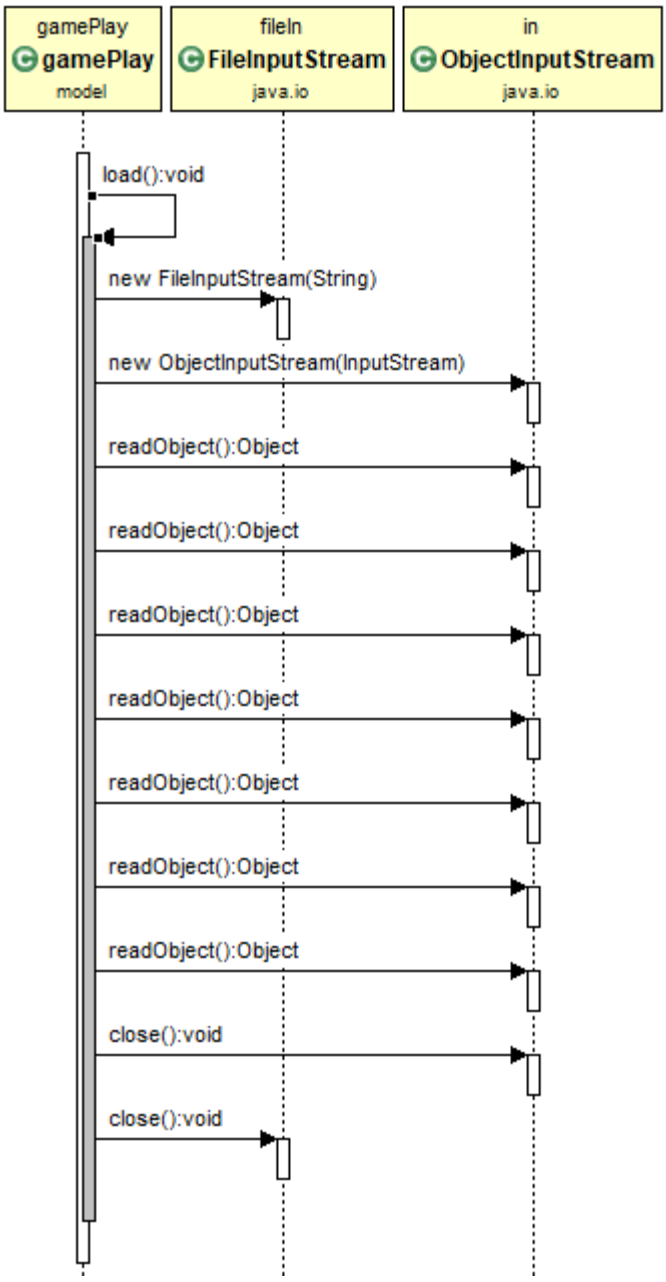


Diagram 10: This sequence diagram shows when load method is called by creating a new file input stream and reading from it.