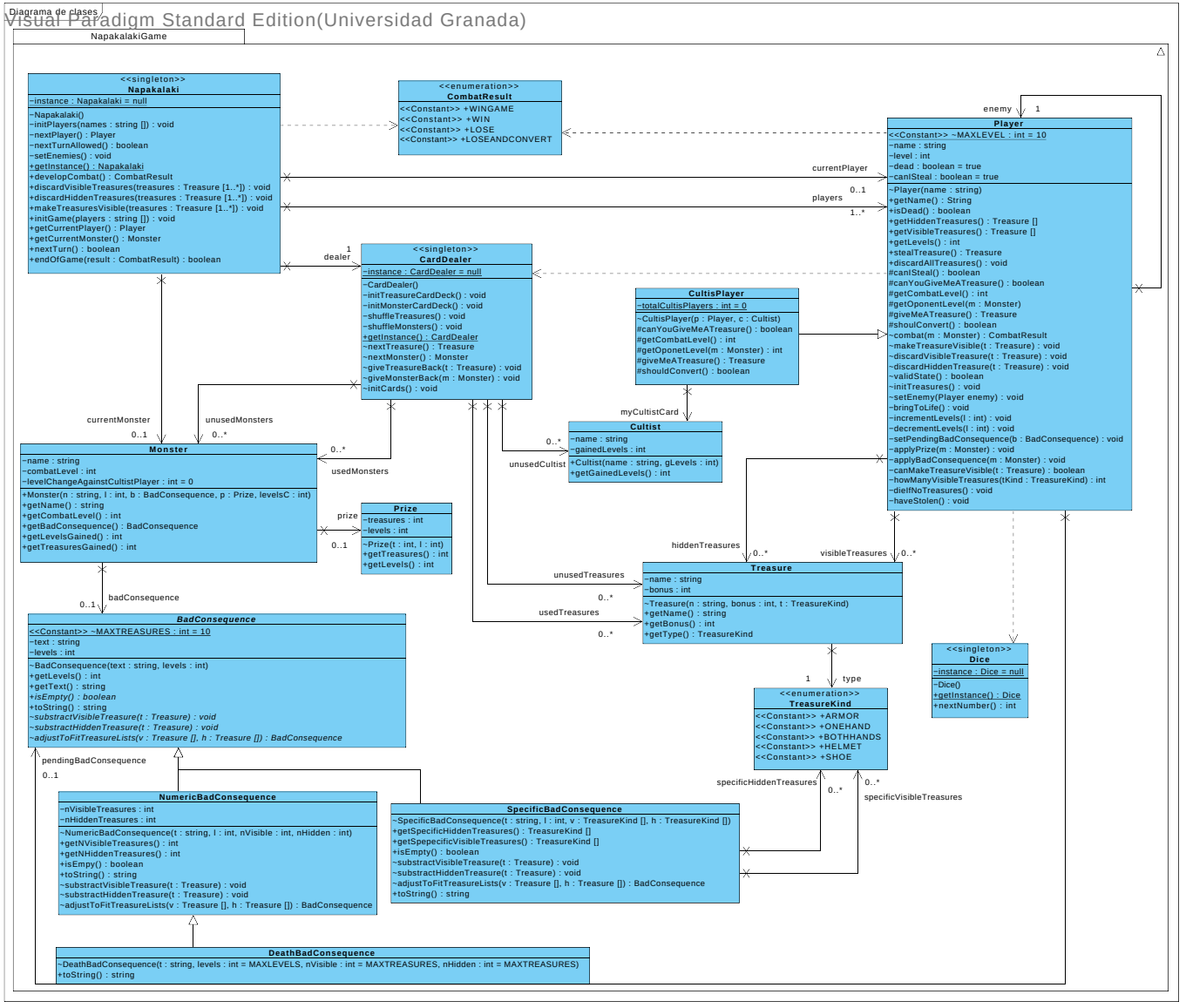
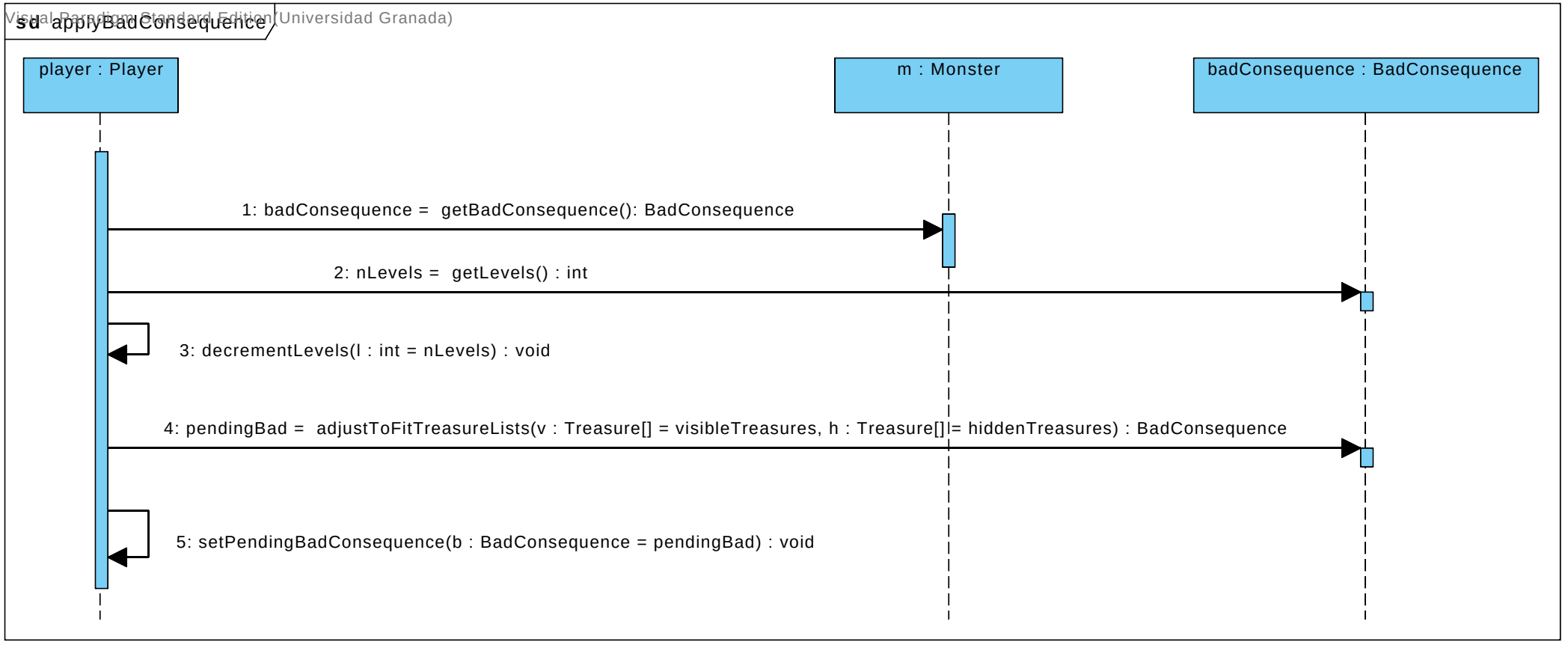
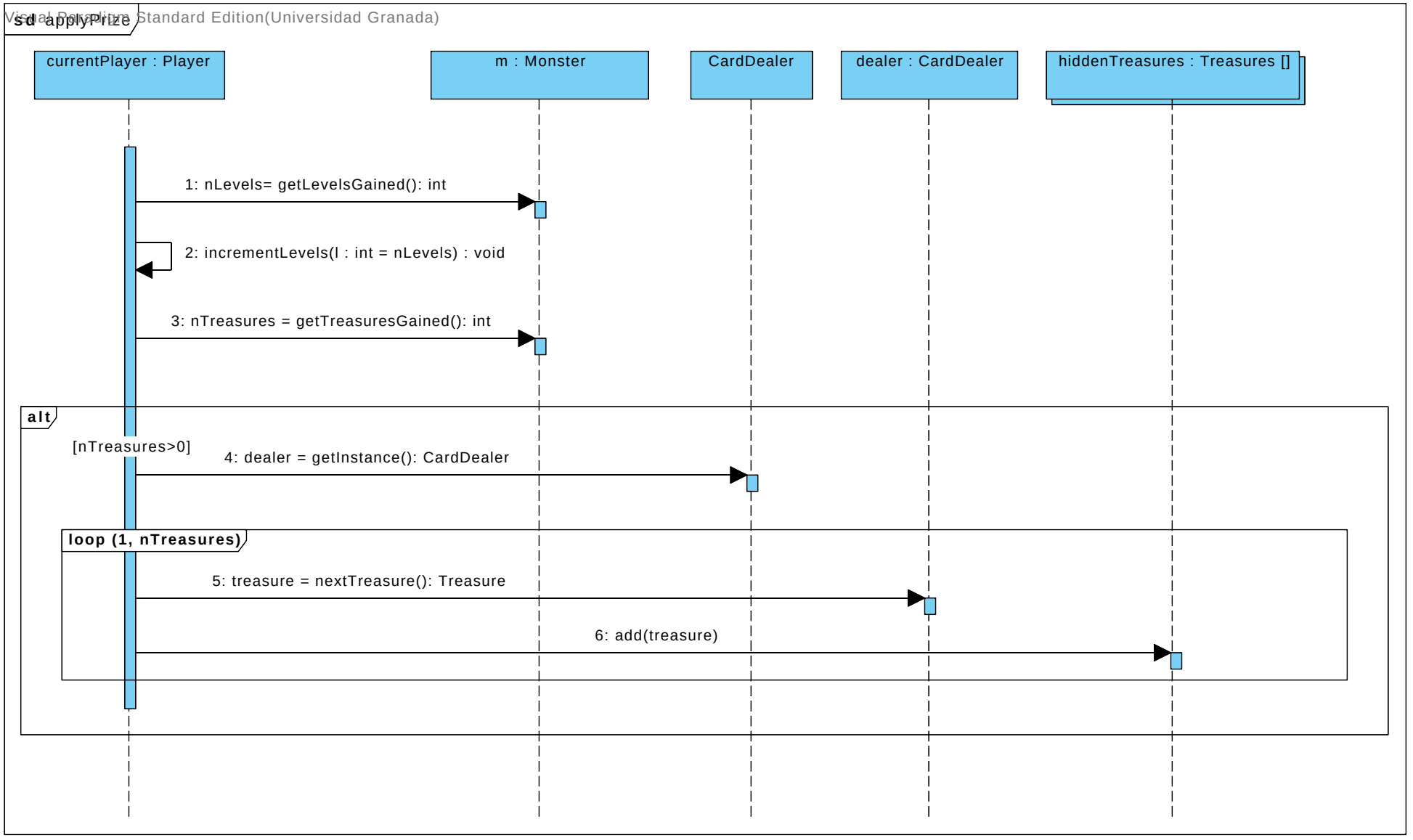
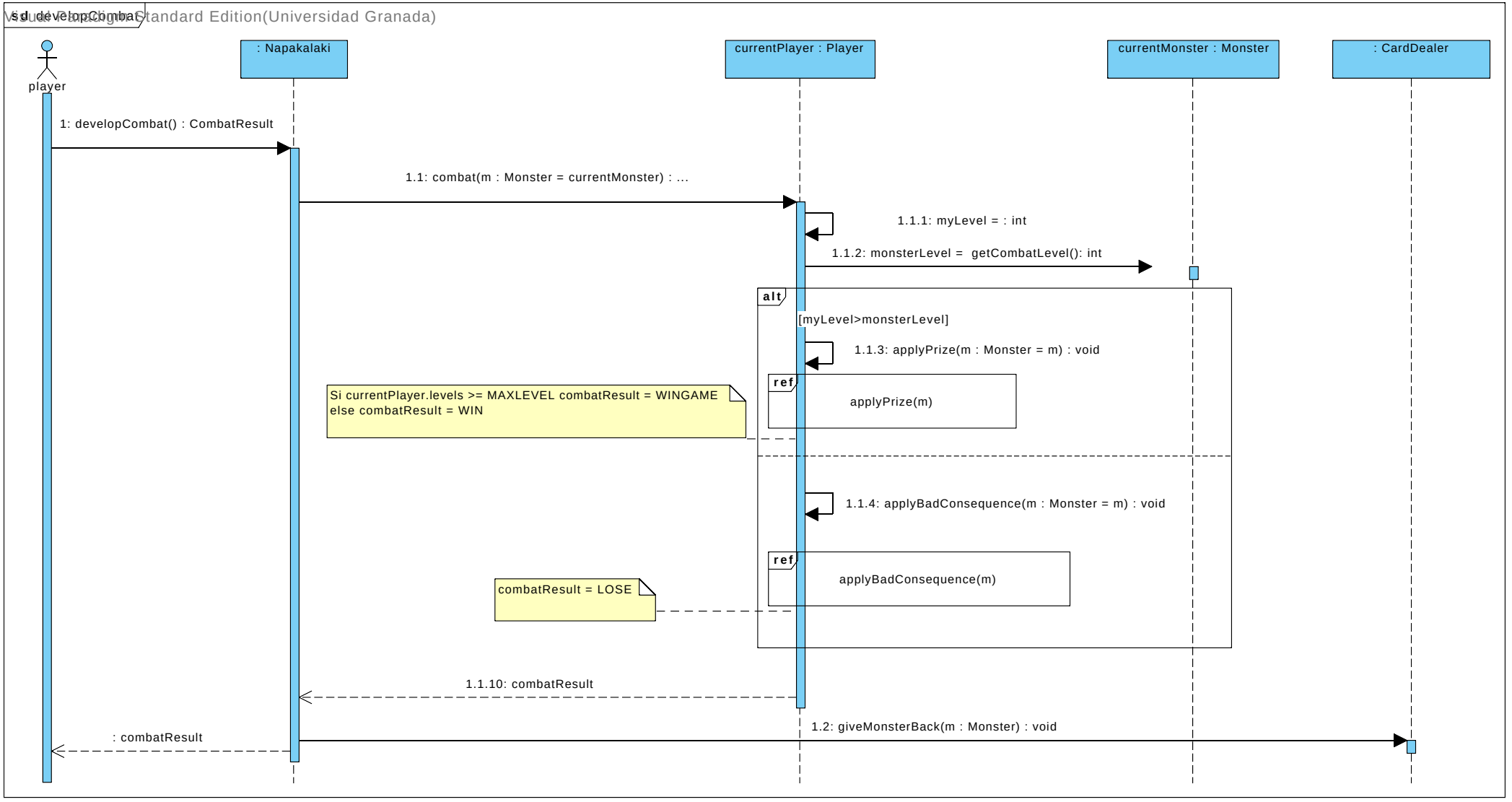
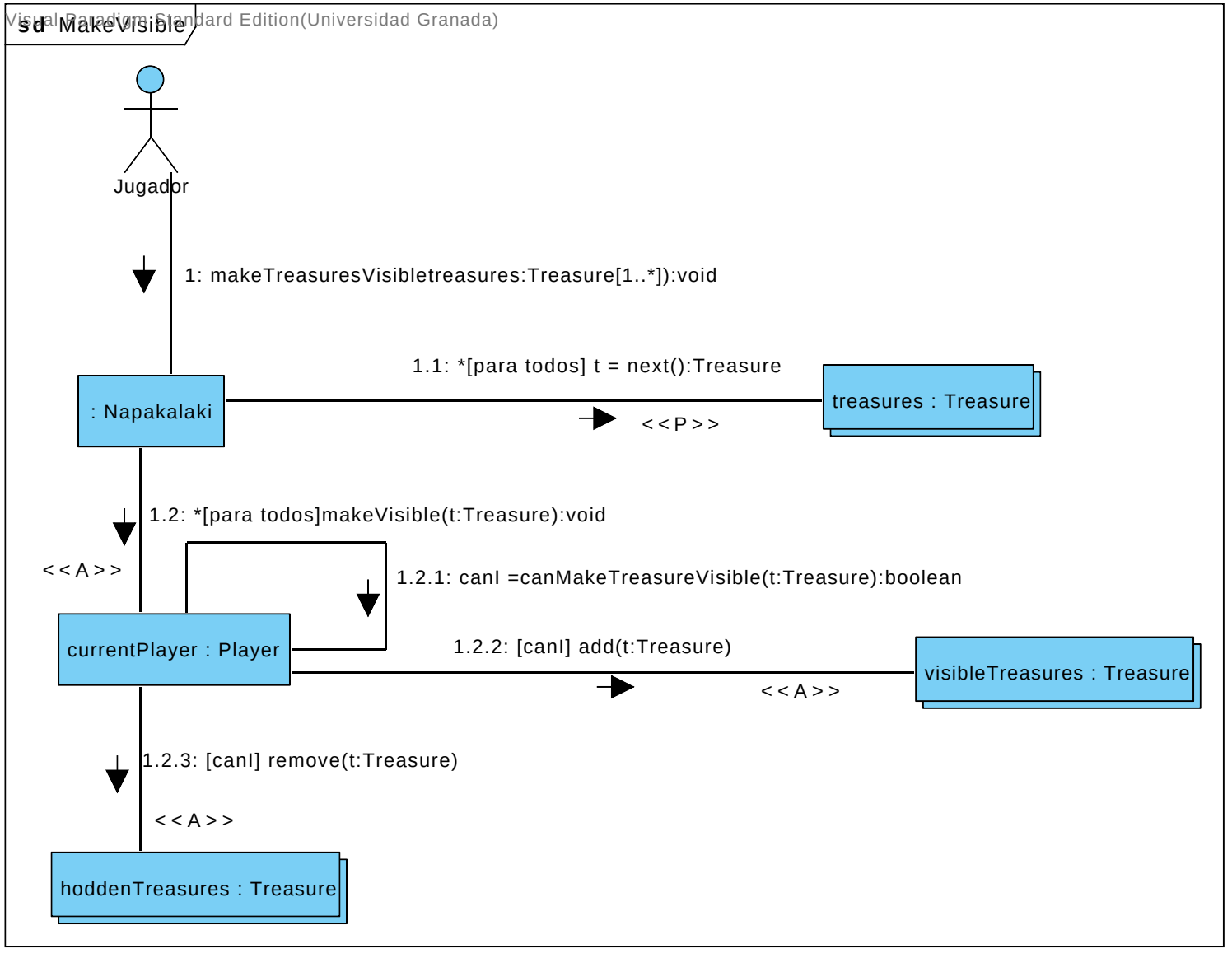


Diagrama de clases
Visual Paradigm Standard Edition(Universidad Granada)
NapakalakiGame
<<singleton>>
Napakalaki
-instance : Napakalaki = null
-Napakalaki()
-initPlayers(names : string []) : void
-nextPlayer() : Player
-nextTurnAllowed() : boolean
-setEnemies() : void
+getInstance() : Napakalaki
+developCombat() : CombatResult
+discardVisibleTreasures(treasures : Treasure [1..*]) : void
+discardHiddenTreasures(treasures : Treasure [1..*]) : void
+makeTreasuresVisible(treasures : Treasure [1..*]) : void
+initGame(players : string []) : void
+getCurrentPlayer() : Player
+getCurrentMonster() : Monster
+nextTurn() : boolean
+endOfGame(result : CombatResult) : boolean
<<enumeration>>
CombatResult
<<Constant>> +WINGAME
<<Constant>> +WIN
<<Constant>> +LOSE
<<Constant>> +LOSEANDCONVERT
Player
<<Constant>> ~MAXLEVEL : int = 10
-name : string
-level : int
-dead : boolean = true
-canISteal : boolean = true
~Player(name : string)
+getName() : String
+isDead() : boolean
+getHiddenTreasures() : Treasure []
+getVisibleTreasures() : Treasure []
+getLevels() : int
+stealTreasure() : Treasure
+discardAllTreasures() : void
#canISteal() : boolean
#canYouGiveMeATreasure() : boolean
#getCombatLevel() : int
#getOponentLevel(m : Monster)
#giveMeATreasure() : Treasure
#shoulConvert() : boolean
~combat(m : Monster) : CombatResult
~makeTreasureVisible(t : Treasure) : void
~discardVisibleTreasure(t : Treasure) : void
~discardHiddenTreasure(t : Treasure) : void
~validState() : boolean
~initTreasures() : void
~setEnemy(Player enemy) : void
~bringToLife() : void
~incrementLevels(l : int) : void
~decrementLevels(l : int) : void
~setPendingBadConsequence(b : BadConsequence) : void
~applyPrize(m : Monster) : void
~applyBadConsequence(m : Monster) : void
~canMakeTreasureVisible(t : Treasure) : boolean
~howManyVisibleTreasures(tKind : TreasureKind) : int
~dieIfNoTreasures() : void
~haveStolen() : void
enemy
1
currentPlayer
0..1
players
1..*
1
dealer
<<singleton>>
CardDealer
-instance : CardDealer = null
~CardDealer()
~initTreasureCardDeck() : void
~initMonsterCardDeck() : void
~shuffleTreasures() : void
~shuffleMonsters() : void
+getInstance() : CardDealer
~nextTreasure() : Treasure
~nextMonster() : Monster
~giveTreasureBack(t : Treasure) : void
~giveMonsterBack(m : Monster) : void
~initCards() : void
CultisPlayer
-totalCultisPlayers : int = 0
~CultisPlayer(p : Player, c : Cultist)
#canYouGiveMeATreasure() : boolean
#getCombatLevel() : int
#getOponetLevel(m : Monster) : int
#giveMeATreasure() : Treasure
#shouldConvert() : boolean
myCultistCard
currentMonster
0..1
unusedMonsters
0..*
Monster
-name : string
-combatLevel : int
-levelChangeAgainstCultistPlayer : int = 0
+Monster(n : string, l : int, b : BadConsequence, p : Prize, levelsC : int)
+getName() : string
+getCombatLevel() : int
+getBadConsequence() : BadConsequence
+getLevelsGained() : int
+getTreasuresGained() : int
0..*
usedMonsters
0..*
unusedCultist
Cultist
-name : string
-gainedLevels : int
+Cultist(name : string, gLevels : int)
+getGainedLevels() : int
prize
0..1
Prize
-treasures : int
-levels : int
~Prize(t : int, l : int)
+getTreasures() : int
+getLevels() : int
hiddenTreasures
0..*
visibleTreasures
0..*
unusedTreasures
0..*
usedTreasures
0..*
Treasure
-name : string
-bonus : int
~Treasure(n : string, bonus : int, t : TreasureKind)
+getName() : string
+getBonus() : int
+getType() : TreasureKind
0..1
badConsequence
BadConsequence
<<Constant>> ~MAXTREASURES : int = 10
-text : string
-levels : int
~BadConsequence(text : string, levels : int)
+getLevels() : int
+getText() : string
+isEmpty() : boolean
+toString() : string
~substractVisibleTreasure(t : Treasure) : void
~substractHiddenTreasure(t : Treasure) : void
~adjustToFitTreasureLists(v : Treasure [], h : Treasure []) : BadConsequence
1
type
<<enumeration>>
TreasureKind
<<Constant>> +ARMOR
<<Constant>> +ONEHAND
<<Constant>> +BOTHHANDS
<<Constant>> +HELMET
<<Constant>> +SHOE
<<singleton>>
Dice
-instance : Dice = null
~Dice()
+getInstance() : Dice
+nextNumber() : int
pendingBadConsequence
0..1
specificHiddenTreasures
0..*
0..*
specificVisibleTreasures
NumericBadConsequence
-nVisibleTreasures : int
-nHiddenTreasures : int
~NumericBadConsequence(t : string, l : int, nVisible : int, nHidden : int)
+getNVisibleTreasures() : int
+getNHiddenTreasures() : int
+isEmpy() : boolean
+toString() : string
~substractVisibleTreasure(t : Treasure) : void
~substractHiddenTreasure(t : Treasure) : void
~adjustToFitTreasureLists(v : Treasure [], h : Treasure []) : BadConsequence
SpecificBadConsequence
~SpecificBadConsequence(t : string, l : int, v : TreasureKind [], h : TreasureKind [])
+getSpecificHiddenTreasures() : TreasureKind []
+getSpepecificVisibleTreasures() : TreasureKind []
+isEmpty() : boolean
~substractVisibleTreasure(t : Treasure) : void
~substractHiddenTreasure(t : Treasure) : void
~adjustToFitTreasureLists(v : Treasure [], h : Treasure []) : BadConsequence
+toString() : string
DeathBadConsequence
~DeathBadConsequence(t : string, levels : int = MAXLEVELS, nVisible : int = MAXTREASURES, nHidden : int = MAXTREASURES)
+toString() : string


Visual Paradigm Standard Edition(Universidad Granada)
sd applyBadConsequence
player : Player
m : Monster
badConsequence : BadConsequence
1: badConsequence = getBadConsequence(): BadConsequence
2: nLevels = getLevels() : int
3: decrementLevels(l : int = nLevels) : void
4: pendingBad = adjustToFitTreasureLists(v : Treasure[] = visibleTreasures, h : Treasure[] = hiddenTreasures) : BadConsequence
5: setPendingBadConsequence(b : BadConsequence = pendingBad) : void


Visual Paradigm Standard Edition(Universidad Granada)
sd applyPrize
currentPlayer : Player
m : Monster
CardDealer
dealer : CardDealer
hiddenTreasures : Treasures []
1: nLevels= getLevelsGained(): int
2: incrementLevels(l : int = nLevels) : void
3: nTreasures = getTreasuresGained(): int
alt
[nTreasures>0]
4: dealer = getInstance(): CardDealer
loop (1, nTreasures)
5: treasure = nextTreasure(): Treasure
6: add(treasure)


sd developCombat
Visual Paradigm Standard Edition(Universidad Granada)
player
: Napakalaki
currentPlayer : Player
currentMonster : Monster
: CardDealer
1: developCombat() : CombatResult
1.1: combat(m : Monster = currentMonster) : ...
1.1.1: myLevel = : int
1.1.2: monsterLevel = getCombatLevel(): int
alt
[myLevel>monsterLevel]
1.1.3: applyPrize(m : Monster = m) : void
ref
applyPrize(m)
Si currentPlayer.levels >= MAXLEVEL combatResult = WINGAME
else combatResult = WIN
1.1.4: applyBadConsequence(m : Monster = m) : void
ref
applyBadConsequence(m)
combatResult = LOSE
1.1.10: combatResult
1.2: giveMonsterBack(m : Monster) : void
: combatResult


Visual Paradigm Standard Edition(Universidad Granada)
sd MakeVisible
Jugador
1: makeTreasuresVisibletreasures:Treasure[1..*]):void
1.1: *[para todos] t = next():Treasure
: Napakalaki
treasures : Treasure
<<P>>
1.2: *[para todos]makeVisible(t:Treasure):void
<<A>>
1.2.1: canI =canMakeTreasureVisible(t:Treasure):boolean
currentPlayer : Player
1.2.2: [canI] add(t:Treasure)
visibleTreasures : Treasure
<<A>>
1.2.3: [canI] remove(t:Treasure)
<<A>>
hoddenTreasures : Treasure