# Weapon (AWeapon)

Post-conditions of Weapon

**Sword (WeaponType)**

sut.Power >= 3

sut.Power <= 6

**Staff (WeaponType)**

sut.Power >= 2

sut.Power <= 4

# Warrior (AWarrior)

Post-conditions of AddWeapon

var sut = new MagicWarrior()

sut.hasWeapon() == false

sut.addWeapon(new Weapon(WeaponType.Staff))

sut.hasWeapon() == true

var sut = new MeleeWarrior();

double meleePowerPre = sut.MeleePower;

var weapon = new Weapon(WeaponType.Sword);

sut.addWeapon(weapon);

sut.MeleePower == meleePowerPre + weapon.Power

var sut = new MagicWarrior();

double magicPowerPre = sut.MagicPower;

var weapon = new Weapon(WeaponType.Staff);

sut.addWeapon(weapon);

sut.MagicPower == magicPowerPre + weapon.Power

Post-conditions of DropWeapon

var sut = new MeleeWarrior();

sut.addWeapon(new Weapon(WeaponType.Sword))

sut.hasWeapon() == true

sut.dropWeapon()

sut.hasWeapon() == false

Post-conditions of Weapon constructor

var sut = new MagicWarrior()

sut.HitPoints == 100

sut.DefensePercentage == 0

# SurrenderAction(ASurrenderAction)

Post-conditions of doAction

var warrior = new MeleeWarrior();

var gameUtil = new GameUtil(null);

var sut = new SurrenderAction(warrior, gameUtil);

var exitStatusPre = Status.Exit;

sut.doAction();

Status.Exit != exitStatusPre

# Position (APosition)

Post-conditions of Equals and GetHashCode

Position position1 = new Position { X = 2, Y = 2 };

Position position2 = new Position { X = 2, Y = 2 };

position1 == position2

position2 = new Position { X = 3, Y = 4 };

position1 != position2

# MeleeWarrior (AMeleeWarrior)

Post-conditions of MeleeWarrior constructor

sut.MeleePower >= 5

sut.MeleePower <= 10

sut.MagicPower >= 1

sut.MagicPower <= 3

# MagicWarrior (AMagicWarrior)

Post-conditions of MagicWarrior constructor

sut.MeleePower >= 1

sut.MeleePower <= 3

sut.MagicPower >= 3

sut.MagicPower <= 8

# GameUtil (AGameUtil)

Post-conditions of createWarrior

var gameUtil = new GameUtil(null);

var sut = gameUtil.createWarrior('M');

sut.GetType() == new MeleeWarrior().GetType();

sut = gameUtil.createWarrior('G');

sut.GetType() == new MagicWarrior().GetType();

// Should not create a warrior for any other character

sut = gameUtil.createWarrior('A');

sut == null

Post-conditions of addWarrior

var sut = new GameUtil(null);

var warriorCountPre = sut.getWarriorsCountForAPlayer(Status.Turn);

sut.addWarrior(sut.createWarrior('M'));

sut.getWarriorsCountForAPlayer(Status.Turn) == warriorCountPre + 1);

Post-conditions of getWarriorsCountForAPlayer

var sut = new GameUtil(null);

var warrior = sut.createWarrior('m');

sut.addWarrior(warrior);

var warriorCountPre = sut.getWarriorsCountForAPlayer(Status.Turn);

sut.deleteWarrior(warrior);

sut.getWarriorsCountForAPlayer(Status.Turn) == warriorCountPre - 1);

Post-conditions of switchTurn

var sut = new GameUtil(null);

PlayerType turnPre = Status.Turn;

sut.switchTurn();

Status.Turn != turnPre);

Status.Turn.GetType() == turnPre.GetType()

Post-conditions of WarriorAttackRange

var board = new Board();

var sut = new GameUtil(board);

var warrior = new MeleeWarrior();

warrior.Position = new Position { X = 0, Y = 0 };

board.ROOMS[0, 1] = 1;

board.ROOMS[1, 1] = 1;

board.ROOMS[1, 2] = 1;

board.ROOMS[2, 0] = 1;

board.ROOMS[2, 2] = 1;

sut.WarriorAttackRange(warrior) == AttackRange.MagicRange;

warrior.Position = new Position { X = 0, Y = 3 };

sut.WarriorAttackRange(warrior) == AttackRange.MagicRange

warrior.Position = new Position { X = 2, Y = 1 };

sut.WarriorAttackRange(warrior) == AttackRange.MeleeRange;

board.ROOMS[0, 2] = 1;

board.ROOMS[2, 3] = 1;

board.ROOMS[4, 0] = 1;

board.ROOMS[4, 2] = 1;

sut.WarriorAttackRange(warrior) == AttackRange.MagicRange;

Post-conditions of findWarriorAtARoom

var sut = new GameUtil(null);

MeleeWarrior meleeWarrior = new MeleeWarrior();

meleeWarrior.Position = new Position { X = 3, Y = 4 };

sut.addWarrior(meleeWarrior);

meleeWarrior = new MeleeWarrior();

meleeWarrior.Position = new Position { X = 2, Y = 1 };

sut.addWarrior(meleeWarrior);

Warrior foundWarrior1 = sut.findWarriorAtARoom(3, 4);

foundWarrior1 != null;

Warrior foundWarrior2 = sut.findWarriorAtARoom(2, 2);

foundWarrior2 == null;

Post-conditions of MoveWarrior

// Should be able to move a warrior

Board board = new Board();

var sut = new GameUtil(board);

var warrior = new MeleeWarrior();

warrior.Position = new Position { X = 2, Y = 2 };

var positionXpre = warrior.Position.X;

var positionYpre = warrior.Position.Y;

sut.MoveWarrior(warrior, Direction.EAST\_NORTH);

warrior.Position.X == positionXpre -1;

warrior.Position.Y == positionYpre + 1;

// Increase defencePercentage

Board board = new Board();

var sut = new GameUtil(board);

var warrior = new MagicWarrior();

warrior.Position = new Position { X = 2, Y = 2 };

double defencePercentagePre = warrior.DefensePercentage;

sut.MoveWarrior(warrior, Direction.EAST);

warrior.DefensePercentage == defencePercentagePre + 0.125;

// Should not increase defencePercentage above 100

Board board = new Board();

var sut = new GameUtil(board);

var warrior = new MagicWarrior();

warrior.Position = new Position { X = 2, Y = 2 };

warrior.DefensePercentage = 99.99;

double defencePercentagePre = warrior.DefensePercentage;

sut.MoveWarrior(warrior, Direction.EAST);

warrior.DefensePercentage == Warrior.MAX\_DEFENSE\_PERCENTAGE;

// Position should be occupied if a warrior is moved there

var board = new Board();

var warrior = new MeleeWarrior();

var sut = new GameUtil(board);

warrior.Position = new Position { X = 2, Y = 2 };

var positionXpre = warrior.Position.X;

var positionYpre = warrior.Position.Y;

sut.MoveWarrior(warrior, Direction.EAST\_NORTH);

board.IsPositionOccupied(new Position { X = positionXpre - 1, Y = positionYpre + 1 }) == true;

Post-conditions of WarriorAttack

// Magic player should be able to attack two places away

var board = new Board();

var sut = new GameUtil(board);

var attackingWarrior = new MagicWarrior();

attackingWarrior.Position = new Position { X = 2, Y = 2 };

var opponentWarrior = new MeleeWarrior();

opponentWarrior.Position = new Position { X = 3, Y = 4 };

sut.addWarrior(attackingWarrior);

sut.addWarrior(opponentWarrior);

sut.WarriorAttack(attackingWarrior, AttackType.MagicAttack, Direction.EAST, Direction.EAST\_SOUTH) == true;

// Melee player should be able to attack two places away

var board = new Board();

var sut = new GameUtil(board);

var attackingWarrior = new MeleeWarrior();

attackingWarrior.Position = new Position { X = 2, Y = 2 };

var opponentWarrior = new MeleeWarrior();

opponentWarrior.Position = new Position { X = 3, Y = 4 };

sut.addWarrior(attackingWarrior);

sut.addWarrior(opponentWarrior);

sut.WarriorAttack(attackingWarrior, AttackType.MagicAttack, Direction.EAST, Direction.EAST\_SOUTH) == true;

// Should not be able to attack outside the board

var board = new Board();

var sut = new GameUtil(board);

var attackingWarrior = new MeleeWarrior();

attackingWarrior.Position = new Position { X = 3, Y = 4 };

sut.addWarrior(attackingWarrior);

sut.WarriorAttack(attackingWarrior, AttackType.MagicAttack, Direction.EAST, Direction.EAST\_SOUTH) == false;

// Should not attack if a warrior does not exist at a position

var board = new Board();

var sut = new GameUtil(board);

var attackingWarrior = new MeleeWarrior();

attackingWarrior.Position = new Position { X = 2, Y = 2 };

var opponentWarrior = new MeleeWarrior();

opponentWarrior.Position = new Position { X = 3, Y = 4 };

sut.addWarrior(attackingWarrior);

sut.addWarrior(opponentWarrior);

sut.WarriorAttack(attackingWarrior, AttackType.MagicAttack, Direction.EAST, Direction.EAST\_NORTH) == false;

// Should be able to attack an opponent

var board = new Board();

var sut = new GameUtil(board);

Warrior warrior = new MeleeWarrior();

warrior.Position = new Position { X = 2, Y = 2 };

var opponent = new MeleeWarrior();

opponent.Position = new Position { X = 2, Y = 3 };

sut.addWarrior(warrior);

sut.addWarrior(opponent);

sut.WarriorAttack(warrior, AttackType.MeleeAttack, Direction.EAST);

sut.findWarriorAtARoom(2, 3) != null;

// Should be able to kill an opponent

var board = new Board();

var sut = new GameUtil(board);

Warrior warrior = new MeleeWarrior();

warrior.Position = new Position { X = 2, Y = 2 };

var opponent = new MeleeWarrior();

opponent.Position = new Position { X = 2, Y = 3 };

opponent.HitPoints = 0.001;

sut.addWarrior(warrior);

sut.addWarrior(opponent);

sut.WarriorAttack(warrior, AttackType.MeleeAttack, Direction.EAST);

sut.findWarriorAtARoom(2, 3) == null;

Post-conditions of GetAddedPowerWithinRange

var board = new Board();

var sut = new GameUtil(board);

Warrior warrior = new MeleeWarrior();

var powerPre = warrior.MeleePower;

sut.GetAddedPowerWithinRange(warrior, 0.5, PowerType.MELEE\_POWER) == powerPre + 0.5;

# Board (ABoard)

Post-conditions of IsPositionOccupied

var sut = new Board();

sut.IsPositionOccupied(2, 4) == false;

sut.ROOMS[2, 4] = 1;

sut.IsPositionOccupied(2, 4) == true;

sut.IsPositionOccupied(new Position { X = 2, Y = 4 }) == true;

Post-conditions of CountBoundaryElements

// Count number of melee range occupied boundary rooms

var sut = new Board();

var position = new Position { X = 3, Y = 3 };

sut.ROOMS[2, 4] = 1;

sut.ROOMS[4, 3] = 1;

sut.CountBoundaryElements(position, 1) == 2;

// Count number of magic range occupied boundary rooms

var sut = new Board();

var position = new Position { X = 3, Y = 3 };

sut.ROOMS[1,1] = 1;

sut.ROOMS[1,4] = 1;

sut.ROOMS[1,5] = 1;

sut.ROOMS[5,1] = 1;

sut.ROOMS[5,4] = 1;

sut.CountBoundaryElements(position, 2) == 5;

// Count occupied boundary rooms near edges in melee range

var sut = new Board();

var position = new Position { X = 3, Y = 5 };

sut.ROOMS[2, 4] = 1;

sut.ROOMS[4, 4] = 1;

sut.ROOMS[4, 5] = 1;

sut.CountBoundaryElements(position, 1) == 3;

// Count occupied boundary rooms near edges in magic range

var sut = new Board();

var position = new Position { X = 0, Y = 0 };

sut.ROOMS[1,2] = 1;

sut.ROOMS[2,0] = 1;

sut.ROOMS[2,2] = 1;

sut.CountBoundaryElements(position, 2) == 3;

Post-conditions of IsPositionInsideBoundary

// Not close position

var sut = new Board();

var currentPositon = new Position { X = 5, Y = 5 };

sut.IsPositionInsideBoundary(currentPositon, Direction.EAST\_SOUTH, Direction.SOUTH) == false;

sut.IsPositionInsideBoundary(currentPositon, Direction.WEST\_NORTH, Direction.WEST) == true;

// Close position

var sut = new Board();

var currentPositon = new Position { X = 5, Y = 5 };

sut.IsPositionInsideBoundary(currentPositon, Direction.EAST\_SOUTH) == false;

sut.IsPositionInsideBoundary(currentPositon, Direction.WEST\_NORTH) == true;

Post-conditions of GetPositionInADirection

// In boundary position

var sut = new Board();

var currentPosition = new Position { X = 2, Y = 2 };

var newPosition = sut.GetPositionInADirection(currentPosition, Direction.EAST);

newPosition == new Position { X = currentPosition.X, Y = currentPosition.Y + 1 };

// Outside boundary position

var sut = new Board();

var currentPosition = new Position { X = 2, Y = 2 };

var newPosition = sut.GetPositionInADirection(currentPosition, Direction.EAST, Direction.EAST\_NORTH);

newPosition == new Position { X = currentPosition.X - 1, Y = currentPosition.Y + 2};

Post-conditions of GetMovementDelta

var sut = new Board();

int preDeltaX = 0;

int preDeltaY = 0;

Position deltaPosition = sut.GetMovementDelta(Direction.EAST\_NORTH);

deltaPosition.X == preDeltaX - 1;

deltaPosition.Y == preDeltaY + 1;

Post-conditions of IsNewPositionOccupied

var sut = new Board();

var currentPosition = new Position { X = 2, Y = 2 };

sut.IsNewPositionOccupied(currentPosition, 0, 2) == false;

sut.ROOMS[2, 4] = 1;

sut.IsNewPositionOccupied(currentPosition, 0, 2) == true;