

**Solution to the**

# **Gaming Parlor**

**Programming Project**

# The Gaming Parlor - Solution

---

## Scenario:

Front desk with dice (*resource units*)

Groups request (e.g., 5) dice (*They request resources*)

Groups must wait, if none available

Dice are returned (*resources are released*)

A list of waiting groups... A “condition” variable

The condition is signalled

The group checks and finds it needs to wait some more

The group (thread) waits

...and goes to the end of the line

## Problem?

# The Gaming Parlor - Solution

---

## Scenario:

Front desk with dice (*resource units*)

Groups request (e.g., 5) dice (*They request resources*)

Groups must wait, if none available

Dice are returned (*resources are released*)

A list of waiting groups... A “condition” variable

The condition is signalled

The group checks and finds it needs to wait some more

The group (thread) waits

...and goes to the end of the line

## Problem?

Starvation!

# The Gaming Parlor - Solution

---

## Approach:

Serve every group “first-come-first-served”.

## Implementation:

Keep the thread at the front of the line separate

“Leader” - the thread that is at the front of the line

Use 2 condition variables.

“Leader” will have at most one waiting thread

“RestOfLine” will have all other waiting threads

# The Threads

---

```
function Group (numDice: int)  
  var i: int  
  for i = 1 to 5  
    gameParlor.Acquire (numDice)  
    currentThread.Yield ()  
    gameParlor.Release (numDice)  
    currentThread.Yield ()  
  endFor  
endFunction
```

```
thA.Init ("A")  
thA.Fork (Group, 4)  
...
```

# The Monitor

---

```
class GameParlor
  superclass Object
  fields
    monitorLock: Mutex
    leader: Condition
    restOfLine: Condition
    numberDiceAvail: int
    numberOfWaitingGroups: int
  methods
    Init ()
    Acquire (numNeeded: int)
    Release (numReturned: int)
    Print (str: String, count: int)
endClass
```

# The Release Method

---

```
method Release (numReturned: int)
  monitorLock.Lock ()

  -- Return the dice
  numberDiceAvail = numberDiceAvail + numReturned

  -- Print
  self.Print ("releases and adds back", numReturned)

  -- Wakeup the first group in line (if any)
  leader.Signal (&monitorLock)

  monitorLock.Unlock ()
endMethod
```

# The Acquire Method

---

```
method Acquire (numNeeded: int)
  monitorLock.Lock ()
  -- Print
  self.Print ("requests", numNeeded)
  -- Indicate that we are waiting for dice.
  numberOfWaitingGroups = numberOfWaitingGroups + 1
  -- If there is a line, then get into it.
  if numberOfWaitingGroups > 1
    restOfLine.Wait (&monitorLock)
  endIf
  -- Now we're at the head of the line. Wait until
  there are enough dice.
  while numberDiceAvail < numNeeded
    leader.Wait (&monitorLock)
  endWhile
  ...
```

# The Acquire Method

---

...

*-- Take our dice.*

```
numberDiceAvail = numberDiceAvail - numNeeded
```

*-- Now we are no longer waiting; wakeup some other  
group and leave.*

```
numberOfWaitingGroups = numberOfWaitingGroups - 1  
restOfLine.Signal (&monitorLock)
```

*-- Print*

```
self.Print ("proceeds with", numNeeded)
```

```
monitorLock.Unlock ()
```

```
endMethod
```