

Modeling and Analysis of a 2D Timed Rescue Game in Maude

Project 목표 : Real-Time Maude로 '여러 구조대원이 제한 시간 안에 여러 위치의 피해자를 구조하는 게임'을 모델링하고, 스케줄링 전략에 따라 데드라인을 만족하는지 LTL 모델체킹으로 분석한다.

규칙 :

1. 맵 : 3x3 격자 (좌표는 (0~2, 0~2))

2. 에이전트

- 플레이어(구조대원): 2명
- 피해자: 2명

3. 행동(Action) 목록

- 이동: 상/하/좌/우 인접 칸으로 한 칸
- 구조 시작: 같은 칸에 있는 피해자를 구조 시작
- 구조 완료: 구조 시작 후 3 time units가 지나면 구조 완료

4. 성공/실패 조건

- 성공: 모든 피해자가 자신의 데드라인 전에 구조 완료되면 win
- 실패: 어떤 피해자라도 데드라인이 지나도록 구조되지 않으면 fail

시간없는 단순 버전 (1D, 구조대원 1명, 피해자 1명) :

• 설계 :

- 일직선 위치 : 0,1,2,3 (1D)
- 구조대원 1명
- 피해자 1명
- 행동 :
 - 오른쪽으로 한 칸 이동
 - 같은 위치면 구조 완료

• Maude :

```
mod RESCUE-SIMPLE is
    protecting NAT . --- 자연수 사용

    --- 위치, 플레이어, 피해자, 전체 상태 정의
    sorts Pos Player Victim State .
```

```
subsort Nat < Pos . --- 위치를 0,1,2,3 자연수로 본다고 가정
```

--- 플레이어 : ID와 위치

```
op player : Nat Pos → Player [ctor] .
```

--- 피해자 : ID, 위치, 구조 여부

```
op victim : Nat Pos Bool → Victim [ctor] .
```

--- 전체 게임 상태 : 시간, 플레이어, 피해자

```
op state : Nat Player Victim → State [ctor] .
```

--- 변수 선언

```
vars T PID X VID VX : Nat .
```

```
var Resc? : Bool .
```

--- 오른쪽으로 한 칸 이동 (위치 < 3 인 경우만)

```
crl [move-right] :
```

```
    state(T, player(PID, X), victim(VID, VX, Resc?))
```

```
⇒
```

```
    state(T, player(PID, s X), victim(VID, VX, Resc?))
```

```
if X < 3 /＼ Resc? == false .
```

--- 구조 (플레이어 위치 = 피해자 위치)

```
rl [rescue] :
```

```
    state(T, player(PID, X), victim(VID, X, false))
```

```
⇒
```

```
    state(T, player(PID, X), victim(VID, X, true)) .
```

```
endm
```

- 실행 결과 :

```
Maude> search [1] state(0, player(0, 0), victim(0, 2, false)) =>* state(T, player(0, 2), victim(0, 2, true)) .
search [1] in RESCUE-SIMPLE : state(0, player(0, 0), victim(0, 2, false)) =>* state(T, player(0, 2), victim(0, 2, true)) .

Solution 1 (state 4)
states: 5  rewrites: 7 in 0ms cpu (0ms real) (148936 rewrites/second)
T --> 0
```

- 승리 조건을 논리식으로 뽑아내기 :

- 무슨 상태를 이긴 상태로 볼까 ? → 피해자가 구조된 상태

```
op goal : State → Bool .
```

```
eq goal(state(T, player(PID, X), victim(VID, VX, Resc?)))
= Resc? . --- 피해자가 구조된 상태면 goal = true
```

- `search`에서 “승리 상태를 만족하는 상태”를 찾자

```

vars S : State .

search [1] state(0, player(0, 0), victim(0, 2, false))
  =>* S:State
    such that goal(S:State) .

```

- 결과 :

```

Maude> search [1] state(0, player(0, 0), victim(0, 2, false)) =>* S:State such that goal(S:State) .
search [1] in RESCUE-SIMPLE : state(0, player(0, 0), victim(0, 2, false)) =>* S:State such that goal(S:State) = true .

Solution 1 (state 4)
states: 5 rewrites: 15 in 0ms cpu (0ms real) (384615 rewrites/second)
S:State --> state(0, player(0, 2), victim(0, 2, true))

```

1단계 : 시간(T) + 데드라인(D) 추가

- 설 :
 - 시간 $T : \text{Nat}$
 - 피해자에 데드라인 $D : \text{Nat}$ 추가
 - 규칙이 한 번 적용될 때마다 T 를 1씩 증가시켜서, “얼마나 많은 행동이 일어났는지”를 시간처럼 사용
 - $\text{deadlineMiss} : \text{State} \rightarrow \text{Bool}$ (데드라인을 넘겼는데 구조가 안 된 상태인지) 함수 추가

- Maude :

```

mod RESCUE-SIMPLE is
  protecting NAT . --- 자연수 사용

  --- 위치, 플레이어, 피해자, 전체 상태 정의
  sorts Pos Player Victim State .

  subsort Nat < Pos . --- 위치를 0,1,2,3 자연수로 본다고 가정

  --- 플레이어 : ID와 위치
  op player : Nat Pos → Player [ctor] .
  --- 피해자 : ID, 위치, 구조 여부
  op victim : Nat Pos Bool → Victim [ctor] .

  --- 전체 게임 상태 : 시간, 플레이어, 피해자
  op state : Nat Player Victim → State [ctor] .

  --- 초기 상태
  --- 플레이어 : 0번, 위치 0 / 피해자 : 0번, 위치 2, 구조 여부 false, 데드라인 5
  op init : → State .
  eq init = state(0, Player(0, 0), victim(0, 2, false, 5)) .

  --- 변수 선언
  vars T PID X VID VX D : Nat .

```

```

var Resc? : Bool .

--- 오른쪽으로 한 칸 이동 (위치 < 3, 아직 구조 전인 경우만 가능)
crl [move-right] :
    state(T, player(PID, X), victim(VID, VX, Resc?, D))
    ⇒
        state(s T, player(PID, s X), victim(VID, VX, Resc?, D))
    if X < 3 /\ Resc? == false .

--- 구조 (플레이어 위치 = 피해자 위치)
rl [rescue] :
    state(T, player(PID, X), victim(VID, X, false, D))
    ⇒
        state(s T, player(PID, X), victim(VID, X, true, D)) .

--- 승리 조건 : 피해자가 구조된 상태
op goal : State → Bool .

eq goal(state(T, player(PID, X), victim(VID, VX, Resc?, D))) = Resc? . --- 피해자가 구조된 상태면 goal = true

--- 데드라인 위반 여부 : 데드라인 초과인데 아직 구조 안 됨
op deadlineMiss : State → Bool .

eq deadlineMiss(state(T, player(PID, X), victim(VID, VX, Resc?, D))) = (Resc? == false) and (T > D) .
endm

```

- 실행 결과 :

```

Maude> red init .
reduce in RESCUE-TIME-SIMPLE : init .
rewrites: 1 in 0ms cpu (0ms real) (1000000 rewrites/second)
result State: state(0, player(0, 0), victim(0, 2, false, 5))
Maude> search [1] init =>* S:State such that goal(S:State) .
search [1] in RESCUE-TIME-SIMPLE : init =>* S:State such that goal(S:State) = true .

Solution 1 (state 4)
states: 5 rewrites: 16 in 0ms cpu (0ms real) (941176 rewrites/second)
S:State --> state(3, player(0, 2), victim(0, 2, true, 5))
Maude> search [1] init =>* S:State such that deadlineMiss(S:State) .
search [1] in RESCUE-TIME-SIMPLE : init =>* S:State such that deadlineMiss(S:State) = true .

No solution.
states: 5 rewrites: 34 in 0ms cpu (0ms real) (311926 rewrites/second)

```

2단계 : 1D에서 여러 플레이어/피해자 추가 (소규모, 2명씩)

- 설계 :
 - 플레이어 2명: P1, P2
 - 피해자 2명: V1, V2

- 상태: state(T, P1, P2, V1, V2)
- 규칙은 “어느 플레이어가 어느 방향으로 움직이는지”를 구분해서 작성
- Maude :

```

mod RESCUE-MULTI-1D is
  protecting NAT .

  sorts Pos Player Victim State .

  subsort Nat < Pos .

  --- 플레이어 : ID와 위치
  op player : Nat Pos → Player [ctor] .

  --- 피해자 : ID, 위치, 구조 여부, 데드라인
  op victim : Nat Pos Bool Nat → Victim [ctor] .

  --- 전체 게임 상태 : 시간, 플레이어 2명, 피해자 2명
  op state : Nat Player Player Victim Victim → State [ctor] .

  --- p 선언 (maude가 못 알아들음)
  op p : Nat → Nat [ctor] .
  eq p(0) = 0 .
  eq p(s N) = N .

  --- 변수 선언
  vars T X1 X2 VX1 VX2 D1 D2 N : Nat .
  vars PID1 PID2 VID1 VID2 : Nat .
  vars R1? R2? : Bool .

  --- 초기 상태
  --- 플레이어 0 : 위치 0
  --- 플레이어 1 : 위치 3
  --- 피해자 0 : 위치 2, 데드라인 6
  --- 피해자 1 : 위치 1, 데드라인 4
  op init : → State .
  eq init = state(0, player(0, 0), player(1, 3), victim(0, 2, false, 6), victim(1, 1, false, 4)) .

  --- P1 이동 (오른쪽) : 아직 두 피해자 모두 구조되기 전이면 이동 가능하다고 가정
  crl [move-right-P1] :
    state(T, player(0, X1), player(1, X2), victim(0, VX1, R1?, D1), victim(1, VX2, R2?, D2))
    ⇒
      state(s T, player(0, s X1), player(1, X2), victim(0, VX1, R1?, D1), victim(1, VX2, R2?, D2))
      if X1 < 3 and (R1? == false or R2? == false) .

  --- P2 이동 (왼쪽)

```

```

crl [move-left-P2] :
    state(T, player(0, X1), player(1, X2), victim(0, VX1, R1?, D1), victim(1, VX2, R2?, D2))
    =>
        state(s T, player(0, X1), player(1, p(X2)), victim(0, VX1, R1?, D1), victim(1, VX2, R2?, D2))
        if X2 > 0 and (R1? == false or R2? == false) .

--- P10| V1 구조
rl [rescue-P1-V1] :
    state(T, player(0, X1), P2:Player, victim(0, X1, false, D1), V2:Victim)
    =>
        state(s T, player(0, X1), P2:Player, victim(0, X1, true, D1), V2:Victim) .

--- P10| V2 구조
rl [rescue-P1-V2] :
    state(T, player(0, X1), P2:Player, V1:Victim, victim(1, X1, false, D2))
    =>
        state(s T, player(0, X1), P2:Player, V1:Victim, victim(1, X1, true, D2)) .

--- P2가 V1 구조
rl [rescue-P2-V1] :
    state(T, P1:Player, player(1, X2), victim(0, X2, false, D1), V2:Victim)
    =>
        state(s T, P1:Player, player(1, X2), victim(0, X2, true, D1), V2:Victim) .

--- P2가 V2 구조
rl [rescue-P2-V2] :
    state(T, P1:Player, player(1, X2), V1:Victim, victim(1, X2, false, D2))
    =>
        state(s T, P1:Player, player(1, X2), V1:Victim, victim(1, X2, true, D2)) .

--- 모든 피해자가 구조되었는지 검사
op allRescued : State → Bool .
eq allRescued(state(T, P1:Player, P2:Player, victim(0, VX1, R1?, D1), victim(0, VX2, R2?, D2))) = (R1? and R2?) .

--- 데드라인 위반 존재 여부
op deadlineMiss : State → Bool .
eq deadlineMiss(state(T, P1:Player, P2:Player, victim(0, VX1, R1?, D1), victim(1, VX2, R2?, D2))) = ((R1? == false and T > D1) or (R2? == false and T > D2)) .
endm

```

- 실행 결과

```

Maude> search [1] init =>* S:State such that allRescued(S:State) .
search [1] in RESCUE-MULTI-1D : init =>* S:State such that allRescued(S:State) = true .

No solution.
states: 54 rewrites: 991 in 1ms cpu (1ms real) (627215 rewrites/second)
Maude> search [1] init =>* S:State such that deadlineMiss(S:State) .
search [1] in RESCUE-MULTI-1D : init =>* S:State such that deadlineMiss(S:State) = true .

Solution 1 (state 32)
states: 33 rewrites: 735 in 0ms cpu (0ms real) (1783980 rewrites/second)
S:State --> state(5, player(0, 3), player(1, 1), victim(0, 2, false, 6), victim(1, 1, false, 4))

```

3단계 : 2D 위치로 확장 (격자 맵)

- 설계 :
 - Pos를 (X, Y) 쌍으로 바꾸고, 이동, 구조, allRescued, deadlineMiss에서 X 또는 Y를 변경하면 됨
- Maude

```

mod RESCUE-MULTI-2D is
  protecting NAT . --- 자연수
  protecting BOOL . --- Bool (true/false 등)

  --- 2D 위치 정의 (X,Y)
  sort Pos .
  op ( _,_ ) : Nat Nat → Pos [ctor] .

  --- 플레이어, 피해자, 전체 상태
  sorts Player Victim State .

  --- 플레이어 : (ID, 위치)
  op player : Nat Pos → Player [ctor] .

  --- 피해자 : (ID, 위치, 구조 여부, 데드라인)
  --- victim(VID, Pos, rescued?, deadline)
  op victim : Nat Pos Bool Nat → Victim [ctor] .

  --- 전체 상태 : (시간 T, 플레이어 2명, 피해자 2명)
  op state : Nat Player Player Victim Victim → State [ctor] .

  --- 맵 크기 (0..maxX, 0..maxY)
  op maxX : → Nat [ctor] .
  op maxY : → Nat [ctor] .
  eq maxX = 2 . --- X = 0,1,2 → 3칸
  eq maxY = 2 . --- Y = 0,1,2 → 3칸

  --- 자연수 감소 함수 (왼쪽/아래 이동용)
  op dec : Nat → Nat .
  eq dec(0) = 0 .
  eq dec(s N) = N .

```

```

--- 변수 선언
vars N : Nat .
vars T D1 D2 : Nat .
vars X1 Y1 X2 Y2 : Nat .
vars VID1 VID2 : Nat .
vars P1 P2 : Player .
vars Pos1 Pos2 : Pos .
vars R1? R2? : Bool .
vars V1 V2 : Victim .
var S : State .

--- 초기 상태
--- 시간: 0
--- 플레이어 0: (0,0)
--- 플레이어 1: (2,2)
--- 피해자 0: (0,2), 데드라인 4
--- 피해자 1: (2,0), 데드라인 5
op init : → State .
eq init =
state(0,
      player(0, (0,0)),
      player(1, (2,2)),
      victim(0, (0,2), false, 4),
      victim(1, (2,0), false, 5)) .

```

--- 이동 규칙 (플레이어 0)

```

--- P1: 오른쪽 이동 (X → X+1)
crl [move-P1-right] :
state(T,
      player(0, (X1, Y1)),
      P2,
      V1,
      V2)
⇒
state(s T,
      player(0, (s X1, Y1)),
      P2,
      V1,
      V2)
if X1 < maxX .

```

```

--- P1: 왼쪽 이동 (X → dec(X))
crl [move-P1-left] :
state(T,
      player(0, (X1, Y1)),

```

```

P2,
V1,
V2)
⇒
state(s T,
    player(0, (dec(X1), Y1)),
P2,
V1,
V2)
if X1 > 0 .

--- P1: 위쪽 이동 (Y → Y+1)
crl [move-P1-up] :
state(T,
    player(0, (X1, Y1)),
P2,
V1,
V2)
⇒
state(s T,
    player(0, (X1, s Y1)),
P2,
V1,
V2)
if Y1 < maxY .

--- P1: 아래쪽 이동 (Y → dec(Y))
crl [move-P1-down] :
state(T,
    player(0, (X1, Y1)),
P2,
V1,
V2)
⇒
state(s T,
    player(0, (X1, dec(Y1))),
P2,
V1,
V2)
if Y1 > 0 .

```

--- 이동 규칙 (플레이어 1)

--- P2: 오른쪽 이동
crl [move-P2-right] :
state(T,

```

P1,
player(1, (X2, Y2)),
V1,
V2)

⇒
state(s T,
P1,
player(1, (s X2, Y2)),
V1,
V2)

if X2 < maxX .

--- P2: 왼쪽 이동
crl [move-P2-left] :
state(T,
P1,
player(1, (X2, Y2)),
V1,
V2)

⇒
state(s T,
P1,
player(1, (dec(X2), Y2)),
V1,
V2)

if X2 > 0 .

--- P2: 위쪽 이동
crl [move-P2-up] :
state(T,
P1,
player(1, (X2, Y2)),
V1,
V2)

⇒
state(s T,
P1,
player(1, (X2, s Y2)),
V1,
V2)

if Y2 < maxY .

--- P2: 아래쪽 이동
crl [move-P2-down] :
state(T,
P1,
player(1, (X2, Y2)),
V1,

```

```

V2)
⇒
state(s T,
P1,
player(1, (X2, dec(Y2))),
V1,
V2)
if Y2 > 0 .

```

--- 구조 규칙 (같은 칸에 있을 때만 구조)

--- P1 이| V1 구조

```

rl [rescue-P1-V1] :
state(T,
player(0, Pos1),
P2,
victim(0, Pos1, false, D1),
V2)
⇒
state(s T,
player(0, Pos1),
P2,
victim(0, Pos1, true, D1),
V2) .

```

--- P1 이| V2 구조

```

rl [rescue-P1-V2] :
state(T,
player(0, Pos1),
P2,
V1,
victim(1, Pos1, false, D2))
⇒
state(s T,
player(0, Pos1),
P2,
V1,
victim(1, Pos1, true, D2)) .

```

--- P2 가 V1 구조

```

rl [rescue-P2-V1] :
state(T,
P1,
player(1, Pos2),
victim(0, Pos2, false, D1),
V2)

```

```

⇒
state(s T,
P1,
player(1, Pos2),
victim(0, Pos2, true, D1),
V2) .

```

--- P2 가 V2 구조
rl [rescue-P2-V2] :

```

state(T,
P1,
player(1, Pos2),
V1,
victim(1, Pos2, false, D2))

⇒
state(s T,
P1,
player(1, Pos2),
V1,
victim(1, Pos2, true, D2)) .

```

--- 분석용 함수들: allRescued / deadlineMiss

--- 모든 피해자가 구조되었는지?
op allRescued : State → Bool .

```

eq allRescued(
state(T,
P1,
P2,
victim(0, Pos1, true, D1),
victim(1, Pos2, true, D2)))
= true .

```

--- 데드라인 위반 상태가 있는지?
op deadlineMiss : State → Bool .

--- 피해자 0이 데드라인 초과인데 아직 구조 안 된 경우
ceq deadlineMiss(
state(T,
P1,
P2,
victim(0, Pos1, false, D1),
V2))
= true
if T > D1 .

```

--- 피해자 1이 데드라인 초과인데 아직 구조 안 된 경우
ceq deadlineMiss(
    state(T,
        P1,
        P2,
        V1,
        victim(1, Pos2, false, D2)))
= true
if T > D2 .

endm

```

- 실행 결과 :

```

Maude> load rescue-multi-2d.maude
Advisory: redefining module RESCUE-MULTI-2D.
Maude> red init .
reduce in RESCUE-MULTI-2D : init .
rewrites: 1 in 0ms cpu (0ms real) (14925 rewrites/second)
result State: state(0, player(0, 0, 0), player(1, 2, 2), victim(0, 0, 2, false, 4), victim(1, 2, 0, false, 5))
Maude> search [1] init =>* S:State such that allRescued(S:State) .
search [1] in RESCUE-MULTI-2D : init =>* S such that allRescued(S) = true .

Solution 1 (state 201)
states: 202 rewrites: 2364 in 4ms cpu (5ms real) (573090 rewrites/second)
S --> state(6, player(0, 2, 0), player(1, 0, 2), victim(0, 0, 2, true, 4), victim(1, 2, 0, true, 5))
Maude> search [1] init =>* S:State such that deadlineMiss(S:State) .
search [1] in RESCUE-MULTI-2D : init =>* S such that deadlineMiss(S) = true .

Solution 1 (state 86)
states: 87 rewrites: 1044 in 3ms cpu (3ms real) (321329 rewrites/second)
S --> state(5, player(0, 1, 0), player(1, 2, 2), victim(0, 0, 2, false, 4), victim(1, 2, 0, false, 5))

```

4단계 : Real-Time Maude 스타일(tick 규칙 + delta/mte)로 업그레이드

- 동기

- 1~3단계까지는 `state(T, P1, P2, V0, V1)` 형태에서 각 이동/구조 규칙이 직접 시간 `T`를 `sT`로 증가시키는 **discrete-time 모델**로 Rescue Game을 명세하였다.
- 이후 교재 17장에서 제시하는 **Real-Time Maude 스타일의 tick 규칙 + delta/mte** 구조를 실제 예제에 적용해 보고자, 기존 모델 위에 실시간 확장을 시도하였다.

- 구조 변화 (모듈 분리)

- 기존 게임 로직은 그대로 `RESCUE-MULTI-2D` 모듈에 두고, 그 위에 별도의 실시간 래퍼 모듈 `RESCUE-RT-MULTI-2D`를 정의하는 구조로 설계하였다.
 - `RESCUE-MULTI-2D`
 - 상태: `State` (`state(T, P1, P2, V0, V1)`)
 - 이동/구조 규칙, `allRescued`, `deadlineMiss` 등의 관찰자 정의
 - `RESCUE-RT-MULTI-2D`
 - `including NAT-TIME-DOMAIN-WITH-INF`

- `protecting RESCUE-MULTI-2D`
- 새로운 구성 정렬과 전역 상태 정의:

```

sort Conf .
subsort State < Conf .
op none : → Conf [ctor] .
op __ : Conf Conf → Conf [ctor assoc comm id: none] .

sort GlobalState .
subsort Conf < GlobalState .
op {} : Conf → GlobalState [ctor] .

```

- 이를 통해 Real-Time Maude에서 사용하는 “구성(Conf) + 전역 상태 `{...}`” 패턴을 Rescue Game에 도입하였다.

- 시간 경과 함수(delta/timeEffect)와 mte 정의 시도

- 교재의 `delta` / `mte` 패턴을 참고하여, 구성에 시간 경과를 적용하는 함수를 다음과 같이 정의하였다.

```

op timeEffect : Conf Time → Conf [frozen (1)] .

eq timeEffect(none, DT) = none .
ceq timeEffect(C1 C2, DT)
= timeEffect(C1, DT) timeEffect(C2, DT)
if C1 /= none and C2 /= none .

eq timeEffect(state(T, P1, P2, V0, V1), DT)
= state(T + DT, P1, P2, V0, V1) .

```

- `mte` (maximum time elapse)는 우선 단순한 형태로 고정하였다.

```

op mte : Conf → Timelinf [frozen (1)] .

eq mte(none) = oo .
ceq mte(C1 C2)
= min(mte(C1), mte(C2))
if C1 /= none and C2 /= none .

eq mte(S:State) = 1 .

```

- 위 정의는 “현재 구성에서 한 번에 헤릴 수 있는 시간은 최대 1”이라는 보수적인 가정을 둔 것으로, 이후 Real-Time 모델링을 정교화할 때 재조정이 필요하다.

- tick 규칙 설계

- Real-Time Maude의 표준 tick 규칙 형식을 따라, 전역 상태에 대해 다음과 같은 규칙을 설계하였다.

```

crl [tick] :
{ C }
⇒
{ timeEffect(C, DT') }
in time DT'
if DT' <= mte(C) .

```

- 의도한 의미는 다음과 같다.
 - 즉시 규칙(기존 RESCUE-MULTI-2D 의 이동/구조 규칙)은 시간을 소모하지 않는 **instantaneous transition**으로 간주한다.
 - 시간의 흐름은 오직 **tick** 규칙을 통해 {C}에서 {timeEffect(C, DT')}로 이동하는 단계에서만 일어나며, 이때 실제 전역 시간은 DT' 만큼 진행된다.
- 현재 discrete-time 모델에서는 이동/구조 규칙이 이미 T 를 증가시키고 있기 때문에, timeEffect 와의 역할 분리를 어떻게 할 것인지(예: State 에서 시간 필드를 제거하고 {T, World} 구조로 완전히 분리할지)는 향후 설계 재검토가 필요한 지점이다.

• 현 단계에서의 한계 및 향후 계획

- 위와 같이 Real-Time Maude 스타일의 **tick** 인프라(timeEffect , mte , tick)를 Rescue Game 위에 올리는 작업을 시도하였으나,
 - 공개되어 있는 **real-time-maude.maude** 가 **Maude 2.x 기반의 예전 버전**이고,
 - 현재 사용 중인 환경은 Maude 3.x라서 oo , in time 와 같은 Real-Time Maude 전용 문법이 제대로 인식되지 않거나, 루프 재초기화 경고가 발생하는 등 환경/버전 호환성 문제를 겪었다.
- 이로 인해 Real-Time Maude 위에서 search ... in time ... 을 이용한 분석까지는 아직 안정적으로 수행하지 못한 상태이며, 실행 가능하고 검증된 부분은 discrete-time 모델과 일반 search 에 기반한 분석까지이다.
- 향후에는
 - 교수님께 권장되는 **Real-Time Maude 파일**(**real-time-maude.maude**)과 **Maude 버전 조합**을 여쭙고,
 - 해당 환경에서 17장을 더 꼼꼼히 학습한 뒤, 현재의 discrete-time Rescue Game 모델을 **tick 규칙 기반 Real-Time 모델로 정제하여 확장**하는 것을 후속 과제로 진행할 계획이다.