United International University

Assignment 02

“ NuSMV implementation”

Group Name: *The Shield*

Members:

#ID01 : 011192047, Md. Abdul Quddus Sagor

#ID02 : 011193058, A.Wahab

#ID03 : 011201262, Azizul Islam Nayem

Course Title: Software Testing & Quality Assurance

Course Code: CSE 4495

Section: B

Instructor: MD. Mohaiminul Islam, Lecturer

Department of Computer Science and Engineering.

<u>Problem Description:</u>

: Suppose you are designing an AI that patrols an area to find enemies and shoot at sight. But you want your AI to be interesting so you try to make it behave somewhat like a human.

If it patrols for more than 30 minutes it gets hungry and looks for food. If it finds an enemy while eating it will drop the snack and immediately start shooting the enemy. Otherwise it will go back to patrolling after its break is finished.

The enemies are also armed sometimes. If your AI sees the enemies have grenades with them the AI will dive for cover. Once it dives for cover it will either shoot or start patrolling again based on enemy visibility.

If your AI kills 20 enemies it will go home for the day (we can consider shooting once kills the enemy). Now take a look at the variables and their possible values –

State: patrol, takeABreak, eatFood, dropSnack, shoot, dive, home

Action: getFood, finishBreak, enemySightedInBreak, enemySighted, noEnemySighted, grenadeSighted

Timer: 0..30 Killed: 0..20

<u>Now in this assignment your job is threefold-</u>

1. Designing the finite state model and implementing it in NuSMV.

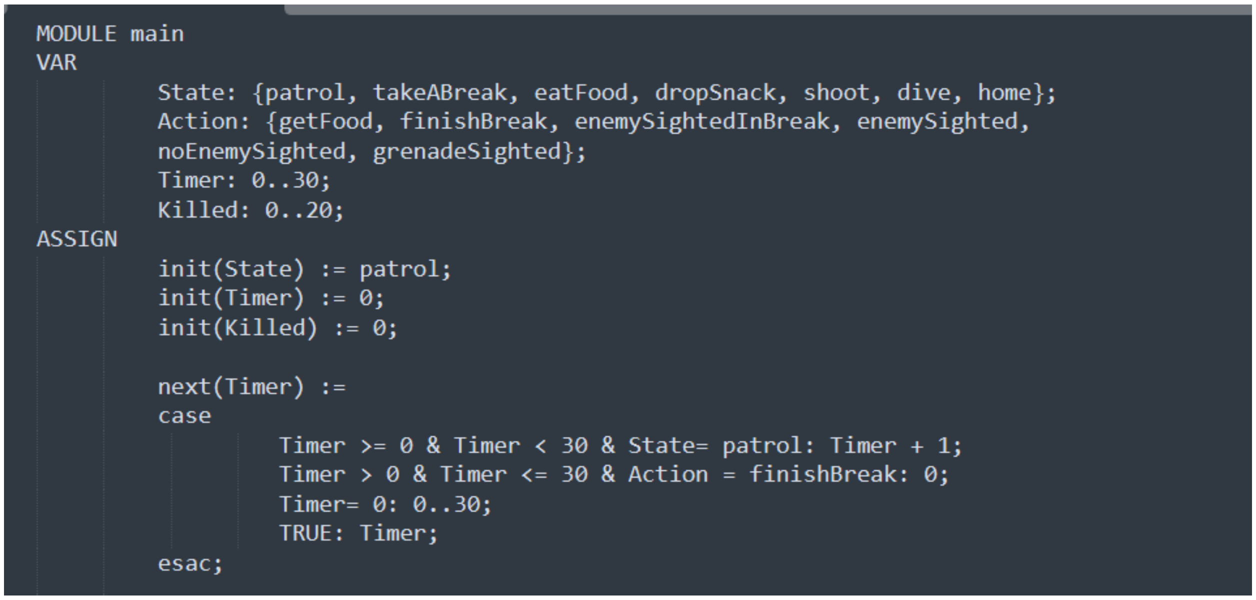
2. Expressing the following informal requirements for this AI in temporal logic (CTL or

LTL).

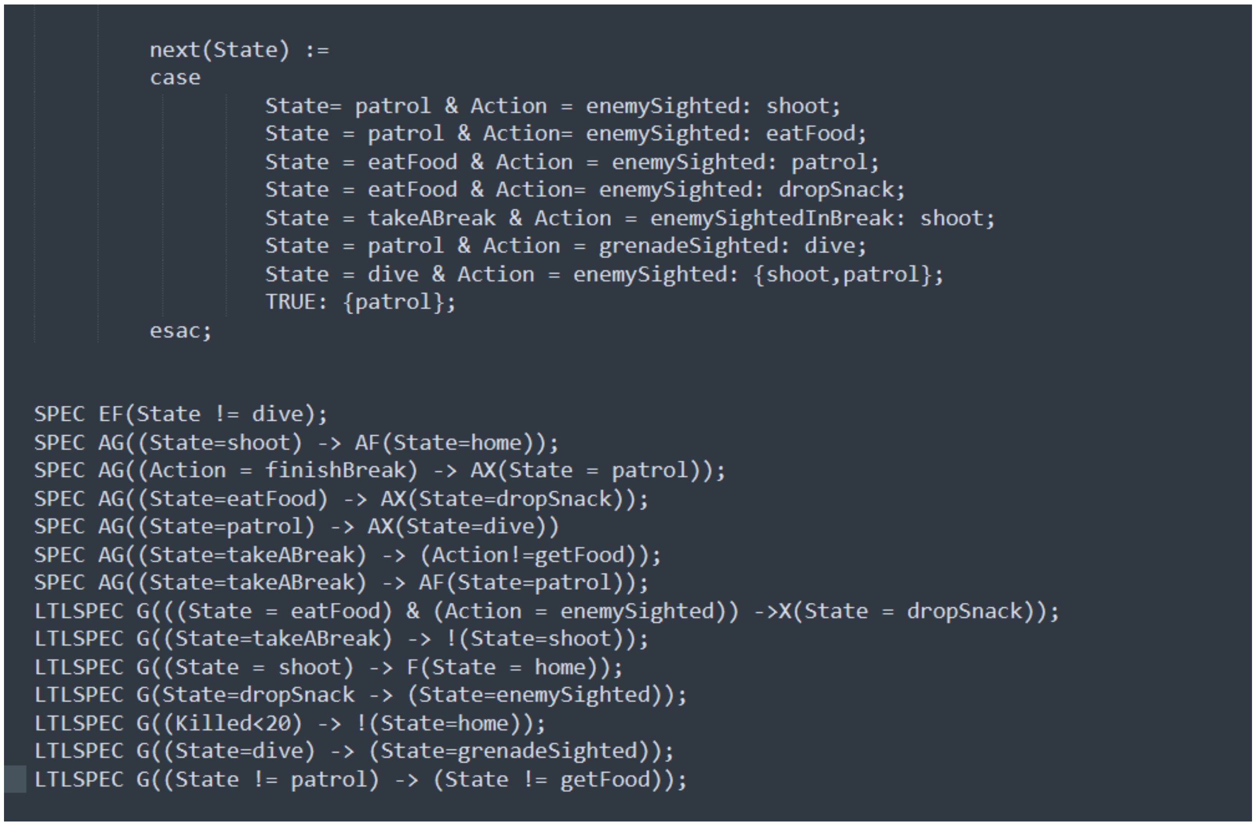
3. Actually checking these if these requirements hold against your implemented NuSMV

model.

<u>Section 1:</u>



```
MODULE main
VAR
        State: {patrol, takeABreak, eatFood, dropSnack, shoot, dive, home};
        Action: {getFood, finishBreak, enemySightedInBreak, enemySighted,
        noEnemySighted, grenadeSighted};
        Timer: 0..30;
        Killed: 0..20;
ASSIGN
        init(State) := patrol;
        init(Timer) := 0;
        init(Killed) := 0;

        next(Timer) :=
        case
                Timer >= 0 & Timer < 30 & State= patrol: Timer + 1;
                Timer > 0 & Timer <= 30 & Action = finishBreak: 0;
                Timer= 0: 0..30;
                TRUE: Timer;
        esac;
```



```
        next(State) :=
        case
                State= patrol & Action = enemySighted: shoot;
                State = patrol & Action= enemySighted: eatFood;
                State = eatFood & Action = enemySighted: patrol;
                State = eatFood & Action= enemySighted: dropSnack;
                State = takeABreak & Action = enemySightedInBreak: shoot;
                State = patrol & Action = grenadeSighted: dive;
                State = dive & Action = enemySighted: {shoot,patrol};
                TRUE: {patrol};
        esac;


SPEC EF(State != dive);
SPEC AG((State=shoot) -> AF(State=home));
SPEC AG((Action = finishBreak) -> AX(State = patrol));
SPEC AG((State=eatFood) -> AX(State=dropSnack));
SPEC AG((State=patrol) -> AX(State=dive))
SPEC AG((State=takeABreak) -> (Action!=getFood));
SPEC AG((State=takeABreak) -> AF(State=patrol));
LTLSPEC G(((State = eatFood) & (Action = enemySighted)) ->X(State = dropSnack));
LTLSPEC G((State=takeABreak) -> !(State=shoot));
LTLSPEC G((State = shoot) -> F(State = home));
LTLSPEC G(State=dropSnack -> (State=enemySighted));
LTLSPEC G((Killed<20) -> !(State=home));
LTLSPEC G((State=dive) -> (State=grenadeSighted));
LTLSPEC G((State != patrol) -> (State != getFood));
```

Section 2:

A list of informal requirements is given below –

o If the AI is eating and it finds an enemy it will drop the snack.

o If the AI starts shooting it will eventually go home.

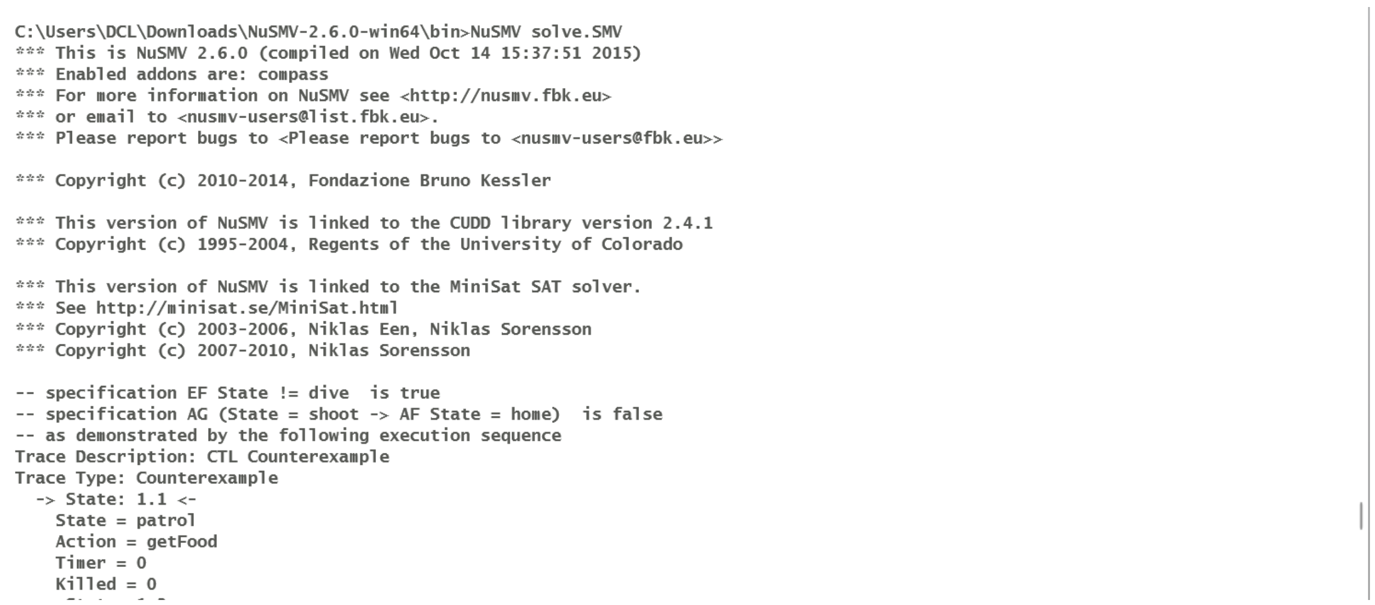
o If the AI isn’t patrolling it will not get hungry.

o The AI might not always dive

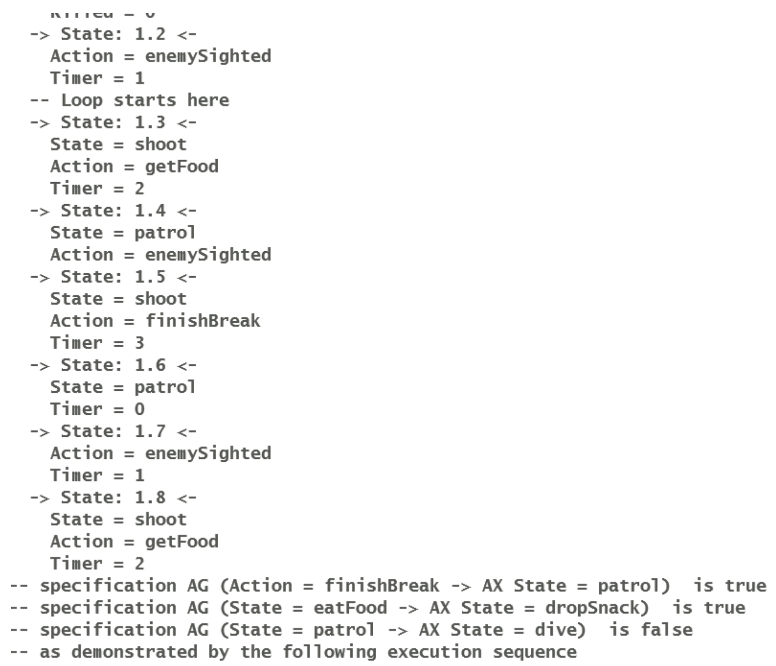
o The AI will always go back to patrolling once its break is finished

- EF(State != dive);
- AG((State=shoot) -> AF(State=home));
- AG((Action = finishBreak) -> AX(State = patrol));
- AG((State=eatFood) -> AX(State=dropSnack));
- AG((State=patrol) -> AX(State=dive))
- AG((State=takeABreak) -> (Action!=getFood));
- AG((State=takeABreak) -> AF(State=patrol));
- G(((State = eatFood) & (Action = enemySighted)) ->X(State = dropSnack));
- G((State=takeABreak) -> !(State=shoot));
- G((State = shoot) -> F(State = home));
- G(State=dropSnack -> (State=enemySighted));
- G((Killed<20) -> !(State=home));
- G((State=dive) -> (State=grenadeSighted));
- G((State != patrol) -> (State != getFood));

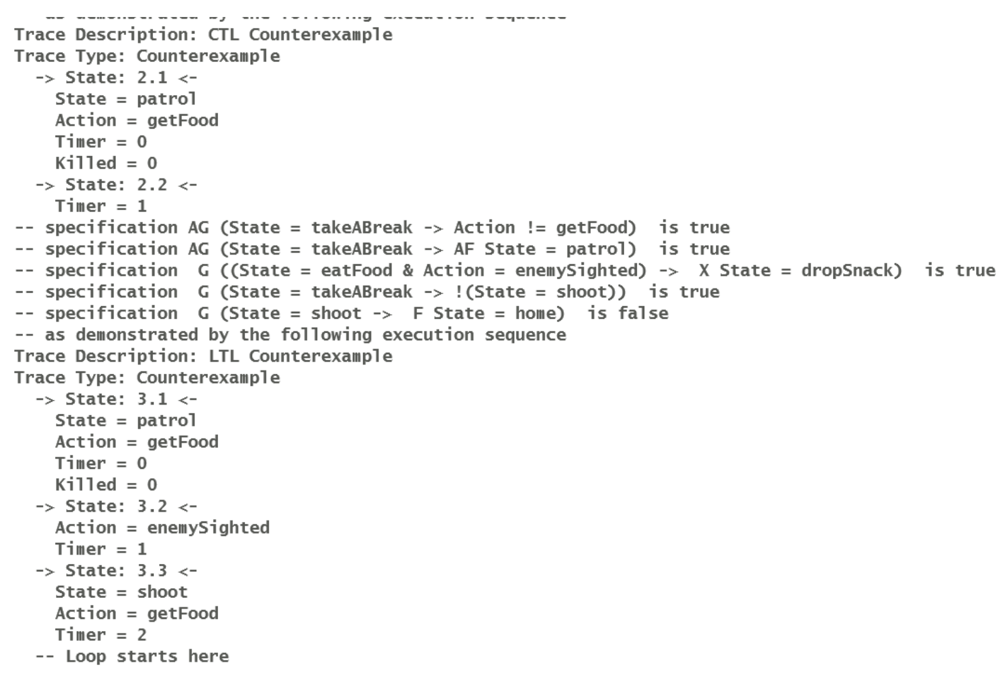
Section 3:



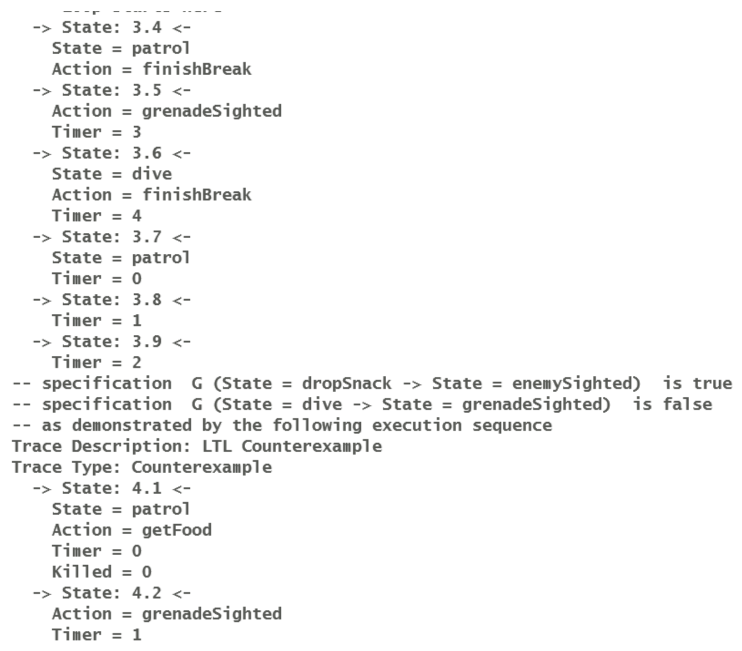
```
C:\Users\DCL\Downloads\NuSMV-2.6.0-win64\bin>NuSMV solve.SMV
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification EF State != dive  is true
-- specification AG (State = shoot -> AF State = home)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    State = patrol
    Action = getFood
    Timer = 0
    Killed = 0
```



```
  -> State: 1.2 <-
    Action = enemySighted
    Timer = 1
  -- Loop starts here
  -> State: 1.3 <-
    State = shoot
    Action = getFood
    Timer = 2
  -> State: 1.4 <-
    State = patrol
    Action = enemySighted
  -> State: 1.5 <-
    State = shoot
    Action = finishBreak
    Timer = 3
  -> State: 1.6 <-
    State = patrol
    Timer = 0
  -> State: 1.7 <-
    Action = enemySighted
    Timer = 1
  -> State: 1.8 <-
    State = shoot
    Action = getFood
    Timer = 2
-- specification AG (Action = finishBreak -> AX State = patrol)  is true
-- specification AG (State = eatFood -> AX State = dropSnack)  is true
-- specification AG (State = patrol -> AX State = dive)  is false
-- as demonstrated by the following execution sequence
```



```
 Trace Description: CTL Counterexample
 Trace Type: Counterexample
   -> State: 2.1 <-
     State = patrol
     Action = getFood
     Timer = 0
     Killed = 0
   -> State: 2.2 <-
     Timer = 1
 -- specification AG (State = takeABreak -> Action != getFood)  is true
 -- specification AG (State = takeABreak -> AF State = patrol)  is true
 -- specification  G ((State = eatFood & Action = enemySighted) ->  X State = dropSnack)  is true
 -- specification  G (State = takeABreak -> !(State = shoot))  is true
 -- specification  G (State = shoot ->  F State = home)  is false
 -- as demonstrated by the following execution sequence
 Trace Description: LTL Counterexample
 Trace Type: Counterexample
   -> State: 3.1 <-
     State = patrol
     Action = getFood
     Timer = 0
     Killed = 0
   -> State: 3.2 <-
     Action = enemySighted
     Timer = 1
   -> State: 3.3 <-
     State = shoot
     Action = getFood
     Timer = 2
   -- Loop starts here
```



```
  -> State: 3.4 <-
    State = patrol
    Action = finishBreak
  -> State: 3.5 <-
    Action = grenadeSighted
    Timer = 3
  -> State: 3.6 <-
    State = dive
    Action = finishBreak
    Timer = 4
  -> State: 3.7 <-
    State = patrol
    Timer = 0
  -> State: 3.8 <-
    Timer = 1
  -> State: 3.9 <-
    Timer = 2
-- specification  G (State = dropSnack -> State = enemySighted)  is true
-- specification  G (State = dive -> State = grenadeSighted)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 4.1 <-
    State = patrol
    Action = getFood
    Timer = 0
    Killed = 0
  -> State: 4.2 <-
    Action = grenadeSighted
    Timer = 1
  -> State: 4.3 <-
    State = dive
    Action = getFood
    Timer = 2
  -- Loop starts here
  -> State: 4.4 <-
    State = patrol
    Action = enemySighted
  -> State: 4.5 <-
    State = shoot
    Action = finishBreak
    Timer = 3
  -> State: 4.6 <-
    State = patrol
    Timer = 0
  -> State: 4.7 <-
    Timer = 1
  -> State: 4.8 <-
    Action = enemySighted
    Timer = 2
-- specification  G (State != patrol -> State != getFood)  is true
-- specification  G (Killed < 20 -> !(State = home))  is true
```

—---------------------------------THE END——--------------------------