THE ADVENTURE GAME A.I.

Chris Tran

Table of Contents

1. Introduction 2
2. The code for the adventure game 2
  2.1. Implement a function to let player use special items........ 2
    a. User interface........ 2
    b. Using a gadget........ 3
    c. Reset adventurer stats after a turn 3
  2.2. The adventurer A.I. 4
    a. Behaviour 1 4
    b. Behaviour 2 5
    c. Behaviour 3........ 6
    d. Behaviour 4 7
    e. Behaviour 5 8
3. Testing........ 8
  3.1. Case 1 8
  3.2. Case 2 8
  3.3. Case 3 9
  3.4. Case 4 9
4. Conclusion........ 9
Appendix 10

1. Introduction

This is a report the final assignment for intelligent systems. The assignment is about an adventure game where player will need to survive in an unknown land. The tasks of the coursework are to extend the adventure game code in CLIPS and implement a program to play the game. The purposes of this report are to document the program and demonstrate some test case that were used to test the program.

2. The code for the adventure game

Full code of the game is in the Appendix.

2.1. Implement a function to let player use special items

a. User interface

```
(defrule choose_special
  (creature (name ~Adventurer))
  (specials $?gadgets)
  (test (>= (length$ $?gadgets) 1))
  (not (just-use ?))
  (turn-counter ?n)
=>
  (printout t "Choose gadget from " $?gadgets " :" crlf
  "[1] Flashgun" crlf
  "[2] Jetpack" crlf
  "[3] Textbook" crlf
  "[4] Nothing" crlf
  )
  (bind ?gadget (read))
  (if (eq ?gadget 1)
   then (assert (use flashgun)))
  (if (eq ?gadget 2)
   then (assert (use jetpack)))
  (if (eq ?gadget 3)
   then (assert (use textbook)))
)
```

The rule *choose_special* will only run when a monster exists, adventurer still have at least one gadget and have not used any item yet. In the next turn, the rule will be revoked again. When the rule is fired, player will need to type in one, two or three to choose a gadget they want to use or four to not use any of the gadget. If no item is chosen, the next rule in the activation queue, which is *choose_action*, will be fired. Otherwise, a fact which show what gadget player want to use will be asserted.

Originally, this rule was supposed to fire after *choose_action* when player want to use item. However, there were some issues when applying it, so the program was changed to run *choose_special* before *choose_action*.

b. Using a gadget

```
(defrule use-gadget
  (declare (salience 190))
  ?adventurer <- (creature (name Adventurer)
    (weapon ?wep)
    (speed ?speed)
    (intelligence ?int)
  )
  ?u <- (use ?gadget)
  ?s <- (specials $?first ?gadget $?last)
=>
  (retract ?u)
  (retract ?s)
  (assert (just-use ?gadget))
  (printout t "You used " ?gadget "." crlf crlf)
  (assert (specials $?first $?last))
  (if (eq ?gadget flashgun)
   then (modify ?adventurer (weapon (+ ?wep 2))))
  (if (eq ?gadget jetpack)
   then (modify ?adventurer (speed (+ ?speed 3))))
  (if (eq ?gadget textbook)
   then (modify ?adventurer (intelligence (+ ?int 2))))
)
```

After a gadget is chosen, this rule will be fired and halt *choose_action* to change player stats depending on which special was used. Priority of 190, which is lower than rule *reset-stats*, was given to the rule in case it is activated. When the rule is fired, player stats will increase depending on the item that was used and a fact which shows what item was used will be asserted.

c. Reset adventurer stats after a turn

```
(defrule reset-stats
  (declare (salience 200))
  ?adventurer <- (creature (name Adventurer)
    (weapon ?wep)
    (speed ?speed)
    (intelligence ?int)
  )
  ?u <- (just-use ?gadget)
  (not (creature (name ~Adventurer)))
=>
  (printout t "Your stats have reset back to normal." crlf crlf)
  (retract ?u)
  (if (eq ?gadget flashgun)
   then
    (modify ?adventurer (weapon (- ?wep 2)))
  )
  (if (eq ?gadget jetpack)
   then
    (modify ?adventurer (speed (- ?speed 3)))
  )
  (if (eq ?gadget textbook)
   then
    (modify ?adventurer (intelligence (- ?int 2)))
  )
)
```

After finishing a turn, if the player is still alive, have used a gadget and there is no creature around, their stats will return to normal. This rule has the highest priority and was written before rule *gen_creature* to force the adventure stats come back to normal before their next battle.

2.2. The adventurer A.I.

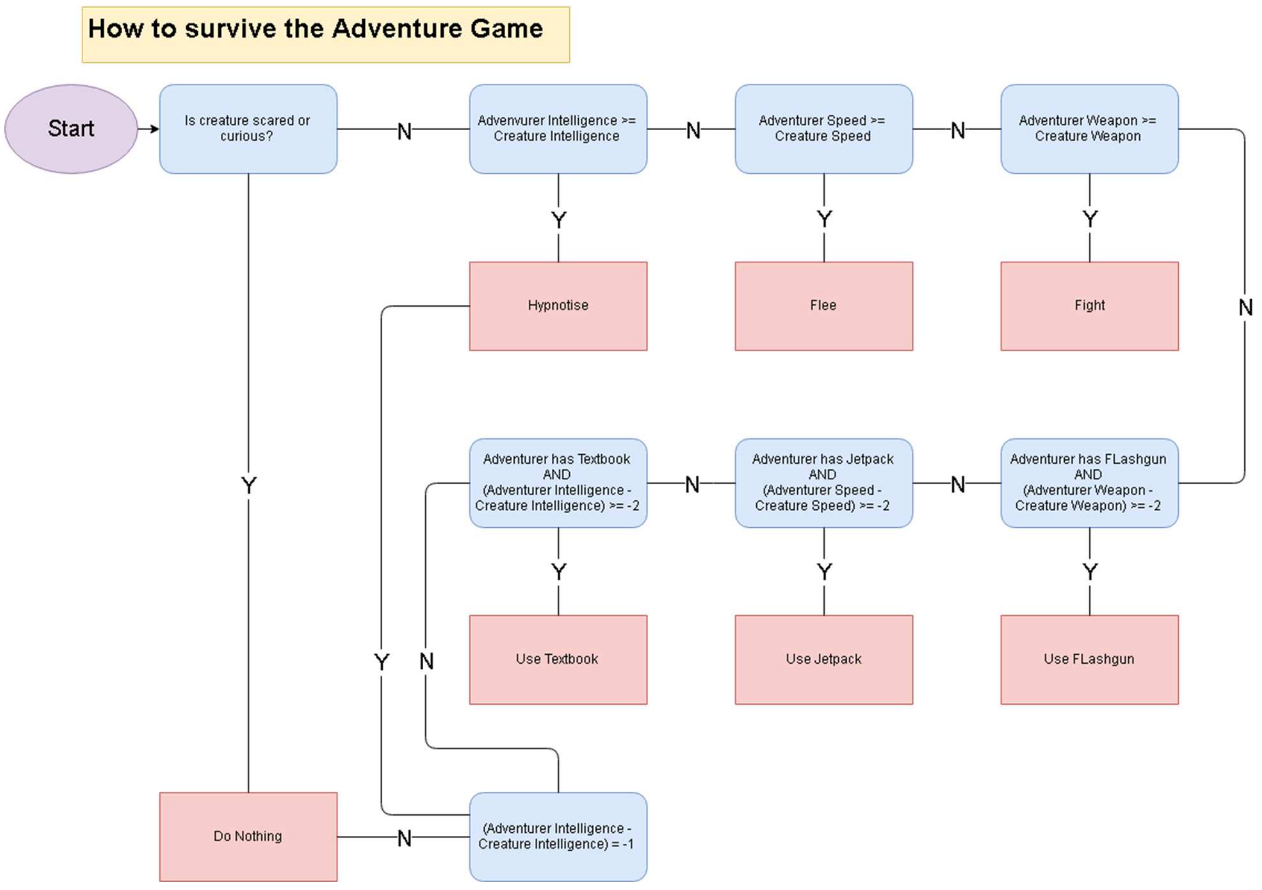
How to survive the Adventure Game
Start
Is creature scared or curious?
N
Advenvurer Intelligence >= Creature Intelligence
N
Adventurer Speed >= Creature Speed
N
Adventurer Weapon >= Creature Weapon
Y
Y
Y
Hypnotise
Flee
Fight
N
Y
Adventurer has Textbook AND (Adventurer Intelligence - Creature Intelligence) >= -2
N
Adventurer has Jetpack AND (Adventurer Speed - Creature Speed) >= -2
N
Adventurer has FLashgun AND (Adventurer Weapon - Creature Weapon) >= -2
Y
Y
Y
Y
N
Use Textbook
Use Jetpack
Use FLashgun
Do Nothing
N
(Adventurer Intelligence - Creature Intelligence) = -1

Figure 1 Adventurer A.I. Decision Tree

a. Behaviour 1

```
(defrule ai-behaviour1
  (creature (name ~Adventurer)
    (attitude scared|curious))
  (not (action ?))  ;; Stop the rule from firing
=>
  (printout t "Behaviour 1: Do nothing." crlf crlf)
  (assert (action do_nothing))
)
```

This rule was written first before any other A.I. behaviour rules to give the program the highest priority to avoid all scared and curious creature. If the pattern of this rule is matched, a fact which is the action to do nothing will be asserted.

b. Behaviour 2

```
;; Rule 1: Hypnotise dumb creature
(defrule ai-behaviour2-rule1
  (declare (salience 180))
  (creature (name Adventurer)
    (intelligence ?aInt))
  (creature (name ~Adventurer)
    (intelligence ?cInt)
    (attitude aggressive))
  (test (>= ?aInt ?cInt))
  (not (action ?))
=>
  (printout t "Behaviour 2 - rule 1: Hypnotise." crlf crlf)
  (assert (action hypnotise))
)

;; Rule 2: Run away from slower creature
(defrule ai-behaviour2-rule2
  (declare (salience 180))
  (creature (name Adventurer)
    (speed ?aSpeed))
  ?c <- (creature (name ~Adventurer)
    (speed ?cSpeed)
    (attitude aggressive))
  (test (>= ?aSpeed ?cSpeed))
  (not (action ?))
=>
  (printout t "Behaviour 2 - rule 2: Flee." crlf crlf)
  (assert (action flee))
)

;; Rule 3: Kill weaker creature
(defrule ai-behaviour2-rule3
  (declare (salience 180))
  (creature (name Adventurer)
    (weapon ?aWeapon))
  (creature (name ~Adventurer)
    (weapon ?cWeapon)
    (attitude aggressive))
  (test (>= ?aWeapon ?cWeapon))
  (not (action ?))
=>
  (printout t "Behaviour 2 - rule 3: Fight." crlf crlf)
  (assert (action fight))
)
```

If the creature is aggressive but can be defeated, this second behaviour will decide which action to take. Within this behaviour, there are three rules. Each rule will compare the different in intelligence, speed and weapon between the adventurer and a creature. If one the adventurer's attribute is higher or equal to the creature, an action will be taken. Intelligence will be checked first then speed and weapon since the highest stat the adventurer has is intelligence and the lowest one is weapon. Priority of 180 is applied to behaviour two to prevent behaviour three, four and five from firing before this behaviour.

c. Behaviour 3

```
;; Rule 1: if the creature can be killed after using flashgun
(defrule ai-behaviour3-rule1
  (declare (salience 150))
  (creature (name Adventurer)
    (weapon ?aWeapon))
  (creature (name ~Adventurer)
    (weapon ?cWeapon)
    (attitude aggressive))
  (test (< ?aWeapon ?cWeapon))
  (test (>= (+ ?aWeapon 2) ?cWeapon))
  (specials $?first flashgun $?last)
  (not (action ?))
  (not (use ?))
=>
  (printout t "Behaviour 3 - rule 1: Use flashgun." crlf crlf)
  (assert (use flashgun))
)

;; Rule 2: if adventurer can run away after using jetpack
(defrule ai-behaviour3-rule2
  (declare (salience 150))
  (creature (name Adventurer)
    (speed ?aSpeed))
  ?c <- (creature (name ~Adventurer)
    (speed ?cSpeed)
    (attitude aggressive))
  (test (< ?aSpeed ?cSpeed))
  (test (>= (+ ?aSpeed 3) ?cSpeed))
  (specials $?first jetpack $?last)
  (not (action ?))
  (not (use ?))
=>
```

```
  (printout t "Behaviour 3 - rule 2: Use jetpack." crlf crlf)
  (assert (use jetpack))
)

;; Rule 3: if the creature can be hypnotised after using textbook
(defrule ai-behaviour3-rule3
  (declare (salience 150))
  (creature (name Adventurer)
    (intelligence ?aInt))
  (creature (name ~Adventurer)
    (intelligence ?cInt)
    (attitude aggressive))
  (test (< ?aInt ?cInt))
  (test (>= (+ ?aInt 2) ?cInt))
  (specials $?first textbook $?last)
  (not (action ?))
  (not (use ?))
=>
  (printout t "Behaviour 3 - rule 3: Use textbook." crlf crlf)
  (assert (use textbook))
)
```

If the aggressive creature cannot be beaten and the adventurer still has some specials, this third behaviour will check if it is possible to defeat the enemy after using an item. Each rule will each the different in attribute between the adventurer and a creature if a gadget is used. If it is possible to beat the creature after using a gadget, a fact showing which item should be use will be asserted. This behaviour will check the attribute weapon first because weapon is the lowest stat the adventurer has. Even after using a special, the adventurer cannot kill a creature will 5 in weapon because the hero only has 4 in weapon. However, if a jetpack or a textbook is used, the explorer will survive. Because of that, this behaviour rule two and three do not need to check if the hero can survive by using an item. However, the rules will still do that in case of a monster that has stats higher than the scale. Priority of 150 is applied to this behaviour to prevent behaviour four and five from firing before this one.

d. Behaviour 4

```
(defrule ai-behaviour4
  (declare (salience 120))
  (creature (name Adventurer)
    (intelligence ?aInt))
  (creature (name ~Adventurer)
    (intelligence ?cInt)
    (attitude aggressive))
  (test (eq (- ?aInt ?cInt) -1))
  (not (action ?))
  (not (specials $?first textbook $?last))
=>
  (printout t "Behaviour 4: Try Hypnotise (50% draw or lose)." crlf crlf)
  (assert (action hypnotise))
)
```

This fourth behaviour will be fired if the creature cannot even be beaten by using a gadget. The program will aim for the 50% draw – lose. Originally there were three rules in this behaviour to use check if it is possible to get 50% draw – lose with weapon attribute. However, that did not increase the chance of survive since the intelligence is the highest stat and the different between the adventure and a very smart creature will be at least -1. Priority of 120 is applied to prevent behaviour 5 from firing.

e. Behaviour 5

```
(defrule ai-behaviour5
  (declare (salience 50))
  (creature (name ~Adventurer)
    (attitude aggressive))
  (not (action ?))
  (not (use ?))
=>
  (printout t "Behaviour 5: Give up." crlf crlf)
  (assert (action do_nothing))
)
```

The fifth behaviour was implemented to if the program cannot decide what action to take and assert do nothing action fact. This behaviour has the lowest priority since it should never be fired in normal situation.

3. Testing

3.1. Case 1

Creature	Status	Weapon	Speed	Intelligence	Expected Result
1	Aggressive	2	5	5	Fight
2	Aggressive	5	5	4	Hypnotise
3	Aggressive	5	3	5	Flee
4	Scared	5	5	5	Do Nothing
5	Curious	1	1	1	Do Nothing

Figure 2 Test case 1 enemies

This test case is used to test behaviour one and two.

3.2. Case 2

Creature	Status	Weapon	Speed	Intelligence	Expected Result
1	Aggressive	4	5	5	Use Gun then Fight
2	Aggressive	5	5	5	Use Jetpack then Flee
3	Aggressive	5	5	5	Use Textbook then Hypnotise

Figure 3 Test case 2 enemies

This test case is used to test behaviour three. This test case is one of the most important test since there was some bug within the behaviour where the program decided to try surviving without using equipment. Hence, some constraint such as the priority and comparing the hero and the creature attribute.

3.3. Case 3

Creature	Status	Weapon	Speed	Intelligence	Expected Result
1	Aggressive	5	5	5	Hypnotise

Figure 4 Test case 3 enemy

The test case is used to test behaviour four. In this test case, the adventurer does not have any gadget left and have to fight a max stats creature. In this case, the program should try to Hypnotise. Like case 2, this behaviour also has some bug and some improvement was implemented.

3.4. Case 4

Creature	Status	Weapon	Speed	Intelligence	Expected Result
1	Aggressive	5	5	5	Hypnotise
2	Aggressive	6	7	7	Do Nothing

Figure 5 Test case 4 enemy

The test case is used to test behaviour five where the hero does not have any special. The test is only to check if this behaviour is working as intended.

4. Conclusion

The program works as designed. After this assignment, I have learnt the basic of CLIPS Expert System and how to implement a simple A.I. into a program.

Appendix

Adventure game and adventure game A.I. code

```
;; Calculate win
(deffunction calculate-win (?ascore ?cscore)
  (bind ?advantage (- ?ascore ?cscore)) ;; difference in scores
  (if (>= ?advantage 2) then (bind ?result win))
  (if (<= ?advantage -2) then (bind ?result loss))
  (if (eq ?advantage 0) then (bind ?result draw))
  (if (eq ?advantage 1) then
    (bind ?luck (mod (random) 2))
  (if (eq ?luck 1) then (bind ?result win)
   else (bind ?result draw)
  )
  )
  (if (eq ?advantage -1) then
    (bind ?luck (mod (random) 2))
  (if (eq ?luck 0) then (bind ?result loss)
   else (bind ?result draw)
  )
  )
  (return ?result)
)

;; Creature template
(deftemplate creature
  (slot name)
  (slot weapon)
  (slot speed)
  (slot intelligence)
  (slot attitude)
)

;; Player Stats and Items
(deffacts adventurer
  (creature
    (name Adventurer)
    (weapon 2)
    (speed 3)
    (intelligence 4)
  )
;; Gadget that should be removed after used
;; Effect   +2weapon +3speed +2int
  (specials flashgun jetpack textbook)
)
```

```
;; Reset stats after using a special
(defrule reset-stats
  (declare (salience 200))
  ?adventurer <- (creature (name Adventurer)
    (weapon ?wep)
    (speed ?speed)
    (intelligence ?int)
  )
  ?u <- (just-use ?gadget)
  (not (creature (name ~Adventurer)))
=>
  (printout t "Your stats have reset back to normal." crlf crlf)
  (retract ?u)
  (if (eq ?gadget flashgun)
   then
    (modify ?adventurer (weapon (- ?wep 2)))
  )
  (if (eq ?gadget jetpack)
   then
    (modify ?adventurer (speed (- ?speed 3)))
  )
  (if (eq ?gadget textbook)
   then
    (modify ?adventurer (intelligence (- ?int 2)))
  )
)

;; Turn
(deffacts game-facts
  (turn-counter 0)
)

;; Generate creature
(defrule gen-creature
  (not (creature (name ~Adventurer)))
  (turn-counter ?n)  ;; updating this will refresh this rule
=>
  (bind ?wep (+ 1 (mod (random) 5)))
  (bind ?speed (+ 1 (mod (random) 5)))
  (bind ?int (+ 1 (mod (random) 5)))
  (bind ?att (+ 1 (mod (random) 3)))
  (if (eq ?att 1)
   then (bind ?atttext scared))
  (if (eq ?att 2)
   then (bind ?atttext curious))
  (if (eq ?att 3)
   then (bind ?atttext aggressive))
  (assert (creature
    (name (random))
    (weapon ?wep)
    (speed ?speed)
    (intelligence ?int)
    (attitude ?atttext)))
;;; The next lines are all for user interface purposes
  (bind ?weptypeno (+ 1 (mod (random) 3)))
;; Weapon description
  (if (eq ?weptypeno 1)
   then (bind ?weptype teeth))
  (if (eq ?weptypeno 2)
   then (bind ?weptype claws))
  (if (eq ?weptypeno 3)
   then (bind ?weptype rocks))
  (if (eq ?wep 1)
   then (bind ?weptext "very blunt"))
  (if (eq ?wep 2)
   then (bind ?weptext "blunt"))
  (if (eq ?wep 3)
   then (bind ?weptext "fairly sharp"))
  (if (eq ?wep 4)
   then (bind ?weptext "sharp"))
  (if (eq ?wep 5)
   then (bind ?weptext "razor sharp"))
;; Speed description
  (if (eq ?speed 1)
   then (bind ?speedtext "very slow"))
  (if (eq ?speed 2)
   then (bind ?speedtext "slow"))
  (if (eq ?speed 3)
   then (bind ?speedtext "fairly fast"))
  (if (eq ?speed 4)
   then (bind ?speedtext "fast"))
  (if (eq ?speed 5)
   then (bind ?speedtext "lightning quick"))
;; Intelligence description
  (if (eq ?int 1)
   then (bind ?inttext "very dumb"))
  (if (eq ?int 2)
   then (bind ?inttext "dumb"))
  (if (eq ?int 3)
   then (bind ?inttext "fairly smart"))
  (if (eq ?int 4)
   then (bind ?inttext "smart"))
  (if (eq ?int 5)
   then (bind ?inttext "really smart"))
;; Monster description
  (printout t "A creature appears!" crlf
              "It has " ?weptext " " ?weptype "." crlf
              "It looks " ?speedtext " and " ?inttext "." crlf
              "It acts in a " ?atttext " manner." crlf crlf)
)

;; Choose gadget
(defrule choose_special
  (creature (name ~Adventurer))
  (specials $?gadgets)
  (test (>= (length$ $?gadgets) 1))
  (not (just-use ?))
  (turn-counter ?n)
=>
  (printout t "Choose gadget from " $?gadgets " :" crlf
  "[1] Flashgun" crlf
  "[2] Jetpack" crlf
  "[3] Textbook" crlf
  "[4] Nothing" crlf
  )
  (bind ?gadget (read))
  (if (eq ?gadget 1)
   then (assert (use flashgun)))
  (if (eq ?gadget 2)
   then (assert (use jetpack)))
  (if (eq ?gadget 3)
   then (assert (use textbook)))
)

;; Use gadget
(defrule use-gadget
  (declare (salience 190))
  ?adventurer <- (creature (name Adventurer)
    (weapon ?wep)
    (speed ?speed)
    (intelligence ?int)
  )
  ?u <- (use ?gadget)
  ?s <- (specials $?first ?gadget $?last)
=>
  (retract ?u)
  (retract ?s)
  (assert (just-use ?gadget))
  (printout t "You used " ?gadget "." crlf crlf)
  (assert (specials $?first $?last))
  (if (eq ?gadget flashgun)
   then (modify ?adventurer (weapon (+ ?wep 2))))
  (if (eq ?gadget jetpack)
   then (modify ?adventurer (speed (+ ?speed 3))))
  (if (eq ?gadget textbook)
   then (modify ?adventurer (intelligence (+ ?int 2))))
)

;; Choose action
(defrule choose-action
  ;; A creature exists which isn't the Adventurer
  (creature (name ~Adventurer))
  (turn-counter ?)
=>
  (printout t "Choose an action [type 1-5]:" crlf
  "[1] Fight" crlf
  "[2] Flee" crlf
  "[3] Hypnotise" crlf
  "[4] Do nothing" crlf
  )
  (bind ?act (read))
  (if (eq ?act 1)
   then (assert (action fight)))
  (if (eq ?act 2)
   then (assert (action flee)))
  (if (eq ?act 3)
   then (assert (action hypnotise)))
  (if (eq ?act 4)
   then (assert (action do_nothing)))
)

;; Fight
(defrule fight
  ?turn <- (turn-counter ?n)
  (creature (name Adventurer)
    (weapon ?aWeapon))
  ?c <- (creature (name ~Adventurer)
    (weapon ?cWeapon))
  ?a <- (action fight)
=>
  (retract ?a)
  (bind ?result (calculate-win ?aWeapon ?cWeapon))
  (if (eq ?result win)
   then (printout t "You killed the creature." crlf crlf))
  (if (eq ?result draw)
   then (printout t "It’s a draw and the creature ran away." crlf crlf))
  (if (eq ?result loss)
   then (printout t "The creature killed you." crlf
                  "You lasted " ?n " turns." crlf crlf)
    (halt))
  (if (neq ?result loss)
   then
    (retract ?turn)
    (retract ?c)
    (assert (turn-counter (+ ?n 1)))
  )
)

;; Flee
(defrule flee
  ?turn <- (turn-counter ?n)
  (creature (name Adventurer)
    (speed ?aSpeed))
  ?c <- (creature (name ~Adventurer)
    (speed ?cSpeed))
  ?a <- (action flee)
=>
  (retract ?a)
  (bind ?result (- ?aSpeed ?cSpeed))
  (if (> ?result 0)
   then (printout t "You ran away." crlf crlf))
  (if (eq ?result 0)
   then (printout t "It’s a draw and the creature ran away." crlf crlf))
  (if (< ?result 0)
   then (printout t "The creature caught and killed you." crlf
                  "You lasted " ?n " turns." crlf crlf)
    (halt))
  (if (>= ?result 0)
   then
    (retract ?turn)
    (retract ?c)
    (assert (turn-counter (+ ?n 1)))
  )
)

;; Hypnotise
(defrule hypnotise
  ?turn <- (turn-counter ?n)
  (creature (name Adventurer)
    (intelligence ?aInt))
  ?c <- (creature (name ~Adventurer)
    (intelligence ?cInt))
  ?a <- (action hypnotise)
=>
  (retract ?a)
  (bind ?result (calculate-win ?aInt ?cInt))
  (if (eq ?result win)
   then (printout t "The creature falls asleep." crlf crlf))
  (if (eq ?result draw)
   then (printout t "It's a draw and the creature runs away." crlf crlf))
  (if (eq ?result loss)
   then (printout t "The creature hypnotised and killed you." crlf
                 "You lasted " ?n " turns." crlf crlf)
    (halt))
  (if (neq ?result loss)
   then
    (retract ?turn)
    (retract ?c)
    (assert (turn-counter (+ ?n 1)))
  )
)

;; Do nothing
(defrule do_nothing
  ?turn <- (turn-counter ?n)
  (creature (name Adventurer))
  ?c <- (creature (name ~Adventurer)
    (attitude ?att))
  ?a <- (action do_nothing)
=>
  (retract ?a)
  (if (eq ?att scared)
   then (printout t "The " ?att " creature ran away." crlf crlf))
  (if (eq ?att curious)
   then (printout t "The " ?att " creature watches you pass by." crlf crlf))
  (if (eq ?att aggressive)
   then (printout t "The " ?att " creature killed you." crlf
                 "You lasted " ?n " turns." crlf crlf)
    (halt))
  (if (neq ?att aggressive)
   then
    (retract ?turn)
    (retract ?c)
    (assert (turn-counter (+ ?n 1)))
  )
)
```

```
;;; AI pattern matching
;; AI behaviour 1
;; Do nothing when fight a scared or curious creature
(defrule ai-behaviour1
  (creature (name ~Adventurer)
    (attitude scared|curious))
  (not (action ?))  ;; Stop the rule from firing
=>
  (printout t "Behaviour 1: Do nothing." crlf crlf)
  (assert (action do_nothing))
)

;; AI behaviour 2: when creature is aggressive but weak
;; Rule 1: Hypnotise dumb creature
(defrule ai-behaviour2-rule1
  (declare (salience 180))
  (creature (name Adventurer)
    (intelligence ?aInt))
  (creature (name ~Adventurer)
    (intelligence ?cInt)
    (attitude aggressive))
  (test (>= ?aInt ?cInt))
  (not (action ?))
=>
  (printout t "Behaviour 2 - rule 1: Hypnotise." crlf crlf)
  (assert (action hypnotise))
)

;; Rule 2: Run away from slower creature
(defrule ai-behaviour2-rule2
  (declare (salience 180))
  (creature (name Adventurer)
    (speed ?aSpeed))
  ?c <- (creature (name ~Adventurer)
    (speed ?cSpeed)
    (attitude aggressive))
  (test (>= ?aSpeed ?cSpeed))
  (not (action ?))
=>
  (printout t "Behaviour 2 - rule 2: Flee." crlf crlf)
  (assert (action flee))
)

;; Rule 3: Kill weaker creature
(defrule ai-behaviour2-rule3
  (declare (salience 180))
  (creature (name Adventurer)
    (weapon ?aWeapon))
  (creature (name ~Adventurer)
    (weapon ?cWeapon)
    (attitude aggressive))
  (test (>= ?aWeapon ?cWeapon))
  (not (action ?))
=>
  (printout t "Behaviour 2 - rule 3: Fight." crlf crlf)
  (assert (action fight))
)

;; AI behaviour 3: when creature is aggressive and strong
;; Rule 1: if the creature can be killed after using flashgun
(defrule ai-behaviour3-rule1
  (declare (salience 150))
  (creature (name Adventurer)
    (weapon ?aWeapon))
  (creature (name ~Adventurer)
    (weapon ?cWeapon)
    (attitude aggressive))
  (test (< ?aWeapon ?cWeapon))
  (test (>= (+ ?aWeapon 2) ?cWeapon))
  (specials $?first flashgun $?last)
  (not (action ?))
  (not (use ?))
=>
  (printout t "Behaviour 3 - rule 1: Use flashgun." crlf crlf)
  (assert (use flashgun))
)

;; Rule 2: if adventurer can run away after using jetpack
(defrule ai-behaviour3-rule2
  (declare (salience 150))
  (creature (name Adventurer)
    (speed ?aSpeed))
  ?c <- (creature (name ~Adventurer)
    (speed ?cSpeed)
    (attitude aggressive))
  (test (< ?aSpeed ?cSpeed))
  (test (>= (+ ?aSpeed 3) ?cSpeed))
  (specials $?first jetpack $?last)
  (not (action ?))
  (not (use ?))
=>
```

```
  (printout t "Behaviour 3 - rule 2: Use jetpack." crlf crlf)
  (assert (use jetpack))
)

;; Rule 3: if the creature can be hypnotised after using textbook
(defrule ai-behaviour3-rule3
  (declare (salience 150))
  (creature (name Adventurer)
    (intelligence ?aInt))
  (creature (name ~Adventurer)
    (intelligence ?cInt)
    (attitude aggressive))
  (test (< ?aInt ?cInt))
  (test (>= (+ ?aInt 2) ?cInt))
  (specials $?first textbook $?last)
  (not (action ?))
  (not (use ?))
=>
  (printout t "Behaviour 3 - rule 3: Use textbook." crlf crlf)
  (assert (use textbook))
)

;; AI behaviour 4: when the aggressive creature is too strong
;;              so the AI will try to get the 50% draw
;;              from -1 score on Int
;;              and try Hypnotise
(defrule ai-behaviour4
  (declare (salience 120))
  (creature (name Adventurer)
    (intelligence ?aInt))
  (creature (name ~Adventurer)
    (intelligence ?cInt)
    (attitude aggressive))
  (test (eq (- ?aInt ?cInt) -1))
  (not (action ?))
  (not (specials $?first textbook $?last))
=>
  (printout t "Behaviour 4: Try Hypnotise (50% draw or lose)." crlf crlf)
  (assert (action hypnotise))
)

;; AI behaviour 5: when there is no solution
;; This behaviour is rarely fired
;; because the game usually stop at behaviour4-rule1
(defrule ai-behaviour5
  (declare (salience 50))
  (creature (name ~Adventurer)
    (attitude aggressive))
  (not (action ?))
  (not (use ?))
=>
  (printout t "Behaviour 5: Give up." crlf crlf)
  (assert (action do_nothing))
)
```