프로그래밍의 원리 2011 가을 - 프로젝트
인셉션

서울대학교 프로그래밍 연구실
이승중, 이영석

2011년 11월 28일

1 프로젝트 목표

수업 시간에 배웠던 프로그래밍의 원리들을 잘 이해하고 적용하여 큰 크기의 프로그램을 작성해 봅니다. 변경된 명세, 공개되지 않은 명세에 잘 적응할 수 있도록 요약을 적절히 사용하여 모듈을 설계하여야 합니다. 또한, 모듈화를 통해 여러 명이 함께 프로그래밍을 하는 방법에 대해서도 익혀 봅니다.

2 프로젝트 명세

2.1 시놉시스

중학생인 효종이는 컴퓨터 게임을 좋아한다. 매일 컴퓨터 게임을 즐기는 효종이는 커서 게임 프로그래머가 되겠다고 다짐했다. 그러던 어느 날 셧다운제의 실행으로 더이상 효종이는 밤에 게임을 즐길 수 없게 되었다. 큰 충격에 빠진 효종이는 어쩔 수 없기 일찍 자게 되지만, 충격이 너무 커서 일까? 잠이 너무 깊게 들어버렸다. 잠에서 깨어나지 못하는 효종이는 어느 인기가수의 노래가사처럼 커서 뭐가 될지 고민하다가 마우스 커서, 아니 튜링머신의 커서가 되어서 큰 세상을 나가지 못하는 안타까운 신세가 되고 말았다. 자 이제 그 꿈속에서 효종이를 구해야 할 때이다.

2.2 목표

튜링머신의 커서가 되어버린 효종이를 잠에서 깨어나는 상태(킥)로 유도합니다. 효종이는 현재 꿈속에서 튜링머신처럼 정해진 룰대로 움직이고 있습니다. 여러분들은 효종이의 꿈속에서 조력자들이 되어 맵을 조작함으로써 킥으로 유도합니다.

2.3 명세

2인 1조가 한 팀이 되어 현재 꿈의 상태, 맵과 룰을 보고 도우미들을 이용하여 테입을 고쳐나갑니다.

- 맵 (테입)

테입 위에는 효종이가 인식할 맵은 비어있거나 심볼이 써져 있습니다. 꿈의 맵은 $n \times 1$ 크기 입니다(프로젝트가 진행됨에 따라 바뀔 수 있습니다) 테입위에 쓸 수 있는 심볼은 다음과 같습니다. (프로젝트가 진행됨에 따라 추가될 수 있습니다.)

- 'p' : 초원
- 'h' : 언덕
- 'f' : 숲
- 's' : 눈

1. 효종이(헤더)는 주어진 룰테이블의 규칙에 따라 움직입니다.
2. 꿈의 타입은 HW7의 튜링머신과 거의 비슷하지만 약간의 추가사항이 있습니다.(도우미)

- 도우미

효종이를 잠에서 깨어나게 도와주는 도우미들이 있습니다. 이들은 각각 능력이 있어서 꿈속에서 역할을 수행할 수 있습니다. (도우미는 프로젝트가 진행됨에 따라 추가될 수 있습니다.) 모든 도우미들은 움직일 수 있습니다.

1. 코브(`Supporter.Cobb`)
   해당 위치의 맵의 지형을 다른 지형으로 바꿀 수 있습니다.(비어있는 땅으로 바꿀 수 없습니다)
2. 사이토(`Supporter.Saito`)
   해당 위치의 맵의 지형을 빈땅으로 바꿀 수 있습니다.

- 규칙

1. 턴제로 움직입니다. 한 턴마다 조력자들은 최대 3번까지 행동(이동, 능력활용)을 할 수 있고, 조력자들의 행동이 끝나면 효종이는 꿈의 상태와 룰을 따라 행동을 합니다.
2. 맵(tape)의 정보는 문자열과 효종이의 위치, 조력자들의 위치 리스트로 구성되어 있습니다.
3. 조력자들은 효종이가 현재 있는 위치의 심볼을 조작할 수 없습니다. (단, 꿈이 초기상태인 경우에는 맵의 지형을 조작할 수 있습니다.)
4. 꿈에는 이미 룰이 정해져있고 꿈에서 깰 때까지 바뀌지 않습니다(프로젝트를 진행하면서 자주 바뀔 수 있는 부분입니다).
5. 맵은 원형 띠처럼 되어있어서 맨 오른쪽 끝에서 오른쪽으로 한번 더 가면 맨 왼쪽 끝으로 가게 됩니다. 반대의 경우도 마찬가지 입니다.
6. 효종이가 현재 꿈의 상태에서 적용할 룰이 없는 경우에는 초기상태로 리셋됩니다.
7. 꿈이 최종상태가 되면 킥을 시도하고, 꿈은 다시 처음 상태가 됩니다.
8. 주어진 턴 안에서 최대한 킥을 시도하는 것이 목표입니다.

2.4 실행 시간 제한

1턴에 조력자들의 행동을 결정하는 시간은 5초입니다. 5초를 초과하면 조력자들은 해당 턴에 아무 일도 하지 않습니다.

3 프로젝트 진행

1. 프로젝트는 무작위로 선정된 2인 1조로 OCaml을 사용하여 진행됩니다.
2. 프로젝트는 서버/클라이언트 방식으로 이루어집니다. 조교는 여러분이 사용할 수 있는 클라이언트의 뼈대 코드를 미리 제공합니다. 서버와 통신하는 기본적인 기능만이 구현되어 있으며 나머지 부분을 구현하는 것은 여러분의 몫입니다.
3. 서버에서는 룰과 매턴의 테입의 모양, 튜링머신의 상태를 클라이언트에게 알려줍니다. 클라이언트는 이 정보를 받아서 조력자들의 행동을 결정합니다.
4. 두번의 제출기회가 주어집니다. 각 제출때마다 프로젝트의 스펙과 진행방식이 변경됩니다.
5. 마지막 제출에 대해서는 수업시간에 진행이 생중계됩니다.
6. 프로젝트 성적은 결과와 보고서 점수의 합으로 매겨집니다.
7. 프로젝트 일정과 명세, 자세한 규칙 및 뼈대코드는 조교 페이지와 게시판을 통해 공개됩니다. 게시판을 자주 확인하시기 바랍니다.

4 모듈 설명

```
(* 효종이가 돌아다닐 맵을 나타냄, 튜링머신의 테입 *)
module Map :
  sig
    type t
    type loc = int       (* 맵에서의 위치를 나타낼때 쓰는 타입 *)
    type symbol =        (* 맵에 쓰여지는 심볼들 *)
        | Blank          (* 아무것도 쓰여있지 않은 상태, 지우면 이 상태가 됨 *)
        | Symbol of char

    (* 맵 타입을 인자로 받아 맵의 크기를 돌려주는 함수 *)
    val size : t -> int

    (* 맵의 한 위치와 맵을 받아 그 위치에 써있는 심볼을 돌려주는 함수 *)
    val get : int -> t -> symbol

    (* 심볼들의 리스트를 받아서 맵을 만들어 주는 함수 *)
```

```
    val make : symbol list -> t
  end

(* 튜링머신의 규칙 테이블을 구성하는 타입들을 모아놓은 모듈 *)
module Ruletable :
  sig
    (* 상태는 문자열 *)
    type state = string

    (* 테입에 대고 할 일 *)
    type todo = Modify of char | Erase | None

    (* 헤드의 움직임 *)
    type move = Left | Right | Stay

    (* 심볼은 맵에서 정의한 심볼을 따른다 *)
    type symbol = Map.symbol

    (* 상태, 심볼, 할일, 헤드움직임, 다음 상태 순으로 이뤄진 튜플의 리스트*)
    type t = (state * symbol * todo * move * state) list
  end

(* 도우미들을 나타내는 모듈 *)
module Supporter :
  sig
    (* 도우미 값의 타입 *)
    type t

    (* 도우미의 종류 *)
    type kind = Cobb | Saito

    (* 도우미 값을 받아서 도우미의 종류를 돌려주는 함수 *)
    val get_kind : t -> kind

    (* 도우미의 현재 위치를 돌려줌 *)
    val get_loc : t -> Map.loc

    (* 도우미의 위치를 설정 *)
    val set_loc : Map.loc -> t -> t

    (* 종류와 위치를 갖고 도우미 값을 만들어냄 *)
    val make : kind -> Map.loc -> t
  end
```

```
(* 클라이언트를 만들기 위해 Client.create를 호출할때 넘는 인자와 관련된 함수, 타
입 *)
module Init_info :
  sig
    type t

    (* Init_info.t 값을 받아 규칙 테이블을 돌려준다 *)
    val ruletable : t -> Ruletable.t

    (* 초반의 도우미들 위치 *)
    val supporters : t -> Supporter.t list

    (* 처음 튜링머신의 상태 *)
    val init_state : t -> Ruletable.state

    (* 튜링머신의 마지막 상태들 *)
    val final_states : t -> Ruletable.state list

    (* Init_info.t 값을 만들어내는 함수 *)
    val make :
      Ruletable.t ->
      Supporter.t list -> Ruletable.state -> Ruletable.state list -> t
  end

(* 매 턴마다 주어지는 상태들 정보 *)
module Info :
  sig
    type t

    (* 현재의 맵 상태 *)
    val map : t -> Map.t

    (* 헤더의 위치 *)
    val header_loc : t -> Map.loc

    (* 도우미들의 상태, 위치정보 *)
    val supporters : t -> Supporter.t list

    (* 몇번째 턴인지 *)
    val turn : t -> int

    (* 현재 튜링머신의 상태 *)
    val state : t -> Ruletable.state
```

```
    (* Info.t 타입을 만들어 주는 함수 *)
    val make :
      Map.t -> Map.loc -> Supporter.t list -> int -> Ruletable.state -> t
  end

(* 클라이언트가 내릴 도우미들의 행동 결정 *)
module Judge :
  sig
    (* 움직일 방향 *)
    type direction = Left | Right

    (* 맵에 쓸 심볼의 타입 *)
    type symbol = char

    (* 도우미의 행동 *)
    type action = Move of direction | Erase | Write of symbol | Stay

    (* 각 도우미들이 수행해야 하는 명령들의 나열 *)
    type t = action list list
  end

(* 클라이언트 모듈이 가져야할 시그니처 *)
module type CLIENT =
  sig
    (* 클라이언트 모듈의 상태를 갖고있는 사용자 정의 데이터 *)
    type t

    (* 프로그램 처음에 한번 불려서 클라이언트를 만들어 낸다 *)
    val create : Init_info.t -> Info.t -> t

    (* 만들어진 클라이언트의 현재 상태를 받고, 매턴마다 오는 정보 둘을 받아 도
우미들이 수행해야 하는 일과 클라이언트가 바뀌어진 상태를 돌려준다 *)
    val judge : Info.t -> t -> Judge.t * t
  end

(* 프로젝트를 직접 구동시키는 부분 *)
module Inception :
  functor (Client : CLIENT) ->
    sig
      (* 초기화 정보들을 받아서 Client를 실행시킨다 *)
      val run : Ruletable.t -> Ruletable.state -> Ruletable.state list -> Map.t -> unit
    end
```