An afternoon at the races

Events portray the activities that go by during a typical afternoon at a hippic center, somewhere at the outskirts of Aveiro. There are four main locations: the *track* where the races take place, the *stable* where the horses rest waiting their turn to enter the competition, the *paddock* where the horses are paraded before the spectators, and the *betting center* where the spectators place their bets on the winning horse.

There are three kinds of intervening entities: the pairs *horse / jockey* participating in the races, the *spectators* watching the races and placing bets on the horse they hope to win and the *broker* who accepts the bets, pays the dividends in case of victory and manages in a general manner all the operations that take place.

K races are run during the afternoon, each with N competitors. M spectators are present. The activities are organized as described below

- the broker announces the next race;
- the participating horses are paraded at the paddock by the jockeys;
- the spectators, after observing the horses and thinking about their winning chances, go to the betting center to place their bet;
- the race takes place and one or more horses are declared winners;
- when somebody wins, he or she goes to the betting center to collect the gains.

At the end of the afternoon, the spectators meet at the bar to have a drink and talk about the events that took place.

Each race is composed of a sequence of position increments of the intervening horse / jockey pairs according to the following rules

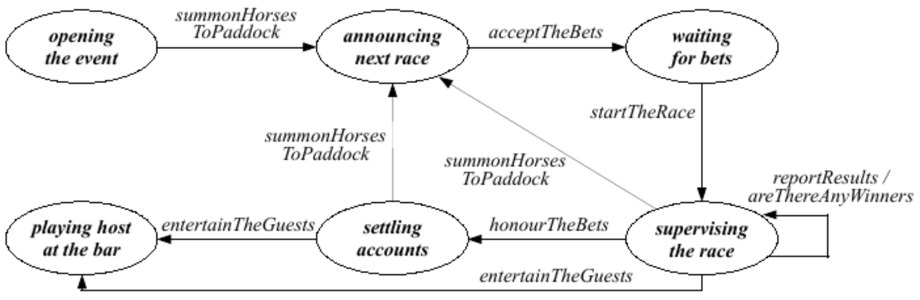
- the track distance for the race k, with k = 0, 1, ... , K-1, is D_k units of length;
- each horse / jockey C_{nk}, with n = 0, 1, ... , N-1 and k = 0, 1, ... , K-1 carries out a single position increment per iteration by moving randomly 1 to P_{nk} length units along its path – the maximum value P_{nk} is specific of a given horse, because they are not all equal, some being more agile and faster than others;
- the horse / jockey pairs move in parallel paths and may be side by side or overtake one another;
- the winner is the pair horse / jockey C_{nk}, with n = 0, 1, ... , N -1 and k = 0, 1, ... , K -1, which, after the completion of an iteration and having overtaken the finishing line, has a position with the highest value;
- in case of a draw, all the horse / jockey pairs with the highest position value are declared winners; the dividends to be received are inversely proportional to their number and rounded to unity.

Assume there are five races, each having four competitors and that the number of spectators is also four. Write a simulation of the life cycle of the horse / jockey pairs, the spectators and the broker using one of the models for *thread* communication and synchronization which have been studied: monitors or semaphores and shared memory.

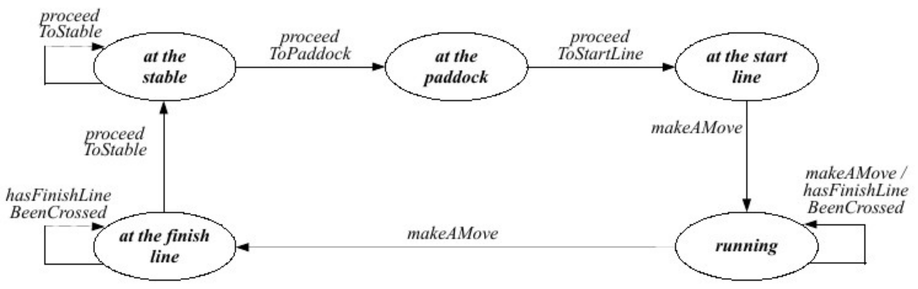
One aims for a distributed solution with multiple information sharing regions, written in Java, run in Linux and which terminates. A *logging* file, that describes the evolution of the internal state of the problem in a clear and precise way, must be included.

Suggestion to solution

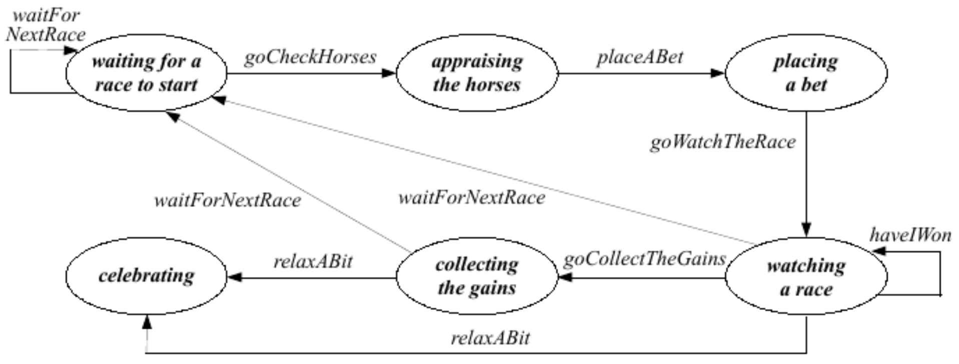
Broker life cycle



Horse / jockey life cycle



Spectator life cycle



Characterization of the interaction

Broker

OPENING_THE_EVENT – initial state (transition)
ANNOUNCING_NEXT_RACE – blocking state
the broker is waken up by the operation *goCheckHorses* of the last spectator to reach the paddock
WAITING_FOR_BETS – blocking state with transition
the broker is waken up by the operation *placeABet* of each of the spectators and blocks again after the bet is accepted; transition only occurs after the betting of all spectators
SUPERVISING_THE_RACE – blocking state
the broker is waken up by the operation *makeAMove* of the last pair horse / jockey crossing the finishing line
SETTLING_ACCOUNTS – blocking state with transition
the broker is waken up by the operation *goCollectTheGains* of each winning spectator and blocks again after honouring the bet; transition only occurs when all spectators have been paid
PLAYING_HOST_AT_THE_BAR – final state (transition)

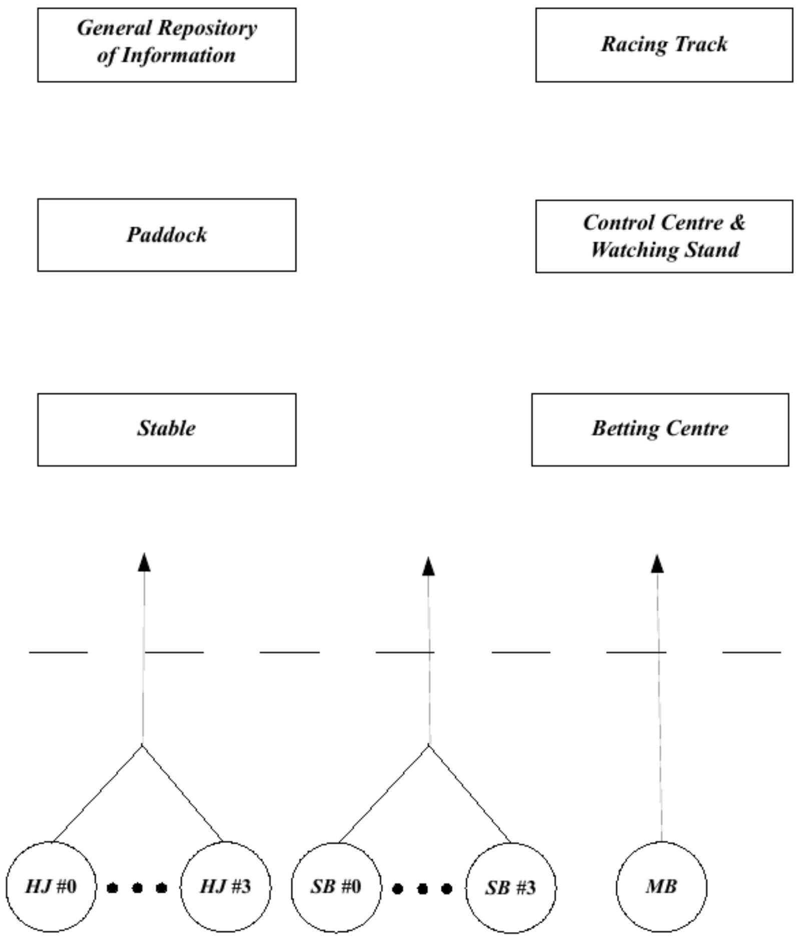
Horse / jockey pair

AT_THE_STABLE – blocking state
the pair horse / jockey is waken up by one of the following operations of the broker: *summonHorsesToPaddock*, during the races, or *entertainTheGuests*, at the end
AT_THE_PADDOCK – blocking state
the pair horse / jockey is waken up by the operation *goCheckHorses* of the last spectator to reach the paddock
AT_THE_START_LINE – blocking state
the pair horse / jockey is waken up by the operation *startTheRace* of the broker (the first) or by the operation *makeAMove* of another horse / jockey pair
RUNNING – blocking state with transition
the pair horse / jockey blocks after carrying out a position increment, unless he crosses the finishing line; he always wakes up the next pair horse / jockey that has not completed the race yet
AT_THE_FINNISH_LINE – transition state

Spectator

WAITING_FOR_A_RACE_TO_START – blocking state
the spectator is waken up by the operation *proceedTo Paddock* of the last pair horse / jockey to reach the paddock
APPRAISING_THE_HORSES – blocking state
the spectator is waken up by the operation *proceedToStartLine* of the last pair horse / jockey to leave the paddock
PLACING_A_BET – blocking state with transition
the spectator blocks in queue while waiting to place the bet; he or she is waken up by the broker when the bet is done
WATCHING_A_RACE – blocking state
the spectator is waken up by the operation *reportResults* of the broker
COLLECTING_THE_GAINS – blocking state with transition
the spectator blocks in queue while waiting to receive the dividends; he or she is waken up by the broker when the transaction is completed
CELEBRATING – final state (transition)

Information sharing regions


General Repository of Information
Racing Track
Paddock
Control Centre & Watching Stand
Stable
Betting Centre
HJ #0
HJ #3
SB #0
SB #3
MB

Guidelines for solution implementation

1. Specify the life cycle and internal properties of each of the *intervening entities*.

2. Specify for each *information sharing region* the internal data structure, the operations which will be invoked, identifying their signature, functionality and who is the calling entity, and the synchronization points.

3. Sketch the *interaction diagram* which describes in a compact, but precise, way the dynamics of your solution. Go back to steps 1 and 2 until you are satisfied the description is correct.

4. Proceed to its coding in Java as specific reference data types.

5. Write the application main program which should instantiate the different *information sharing regions* and the different *intervening entities*, then start the different entities and finally wait for their termination.

6. Validate your solution by taking several runs and checking for each, through the detailed inspection of the logging file, that the output data is indeed correct.