Quidditch in Prolog - *Reasoning and Logic for Rule Based Computer Games*

Quidditch is a fictional game played between two teams in the Harry Potter series of novels by J.K. Rowling (those unfamiliar wth the concept can familiarize themselves with the concept here http://en.wikipedia.org/wiki/Quidditch and http://harrypotter.wikia.com/wiki/Quidditch).

A software company is developing a Quidditch game for which the reasoning and logic software component will be implemented in Prolog. This component will monitor the application of the various rules during game-play to ensure compliance. It will then inform the main software engine of the game (that does graphical display etc.) when the conditions of certain logic conditions have been met and hence that the rule outcome needs to be applied within the game (e.g. foul, goal etc.). Prolog is uniquely suited for such logic and reasoning tasks. Other aspects of game-play such as the generation of player actions and the graphical user interface will be handled by other software components (not implemented here).

As a software engineer you are tasked with developing Prolog predicates that will implement the set of knowledge, rules and game logic outlined above. *In the absence of a definitive and open-access source of Quidditch rules, the company will make use of those stated as quoted below from the above URLs*. This assignment will guide you through the various stages of this task.

Part A - Knowledge Representation

A basic knowledge representation is provided for the game in file *base.pl.* You will need to load *base.pl* in order to use this knowledge base. This defines the dimensions of the pitch (pitch/4), the positions of the goals (goal/4), the ball (ball/1) and the teams (team/2) as prolog predicates. Furthermore it defines the notion of a location on or off the pitch (location/3) that a given game entity (ball or player) may be at (valid_position/4).

Quidditch is played on a pitch defined in 3D space that for the purposes of the game we will assume is rectangular. Game entities (balls or players) may be present at any valid location (location/3) on or off the pitch during game-play at a 3D integer location, (X,Y, Z). Two entities are defined as interacting if they are at the same location (e.g. two players with same location collide, a player and a ball at the same location results in the player catching the ball (for the purposes of the game). Change in player and ball positions and the interpolation of "flight paths" as they move around the pitch will be computed as a series of the player and ball locations by the main game engine itself – ignore this part of the problem here.

At any given point in time, player positions are defined by knowledge base predicates player/4 and similarly ball positions via ball/4 – an example is provided in.*base.pl* (this is essentially the game state against which your predicates are evaluated). Similarly the current score for a given team is provided by score/2. These predicates can be dynamically updated during game play using the assert/1 and delete/1 Prolog predicates (see lecture 9 – however, we will not be using this functionality here).

Based on the provided knowledge base in *base.pl,* define the following predicates in a file named PartA.pl.

A1. Define the predicate player(P, Team) (player/2) required by player_position/4 that holds if

and only if player P is a valid member of a defined team (see *base.pl, valid_position(P, X,Y,Z))*.

A2. Define a predicate on_pitch(X,Y,Z) that holds if and only if (X,Y,Z) is a valid location in the game and is inside the pitch boundary as defined by predicate pitch/4.

A3. Define a corresponding predicate off_pitch(X,Y,Z) that holds if and only if (X,Y,Z) is not a valid location inside the pitch boundary as defined by predicate pitch/4.

A4. Define valid_team(T) that holds if and only if the members of team T are fielding a valid team in terms of size and formation - *"Each team is made up of seven players, consisting of three Chasers, two Beaters, one Keeper and one Seeker."*

Once completed, uncomment the Part A loading part in your base file (*base.pl*) so that these predicates are automatically loaded from *base.pl.*

In order to test part C you will need to define a number a number of game configurations using player/4 and ball/4 (for player and ball positions). Put these test examples (and results) in your separate prolog file *game.pl* such that you can comments in and out different game configurations for testing. Clearly comment the tests that apply to Part A.

Part B – Game Rule Representation and Reasoning

Based on the provided knowledge base in *base.pl,* and in your answer to Part A. *(remember: to interact any two entities must be in the same location – this is the only criterion)*

B1. In quidditch, *" No other player aside from the Seeker is allowed to touch the Snitch (ball)"* . Define a predicate snitch_caught(T) that holds if and only if the correct player of team T has caught the snitch (by being in the same location as it).

B2. In quidditch, a goal is scored *"by sending the red, football-sized Quaffle (ball) through any of the three goal hoops"*. Define a predicate goal_scored(T) that holds if and only if the position of the ball (ball/4) is within radius, R (in the Y - Z plane), of the centre of the goal that does not belong to team T (i.e. the opposing team's goal).

B3. In quidditch a foul can be committed by a team, based on the actions of one of their players under a number of disjunctive circumstances (as follows):-

- *"Players must not stray over the boundary lines of the pitch, although they may fly as high as desired."*
- *" No player may seize any part of an opponent's broom to slow or hinder the player … or …. collide. " or "No player may knock the opposing team's Keeper out of the way so they can score a goal easily." etc*. - basically any form of inter-player contact.
- Players *"must not still be in contact with the Quaffle (ball) as it passes through a hoop —the Quaffle must be thrown through."*

- *"No player other than the Seeker may touch or catch the Golden Snitch.(ball)"* (note that this is different from B.1)

Define a predicate foul(T) that holds if and only if a player from team T commits a foul (as outlined above) based on the current state of the game.

Once completed, uncomment the Part B loading part in your base file (*base.pl*) so that these predicates are automatically loaded from *base.pl.*

Again in order to test part B you will need to define a number of game configurations using player/4 and ball/4 (for player and ball positions). Add these test examples (and results) into your separate prolog file *game.pl* such that you can comment in and out different game configurations for testing. Clearly comment the tests that apply to Part B.

Part C – Game Play Logic

Based on the provided knowledge base in *base.pl,* and in your answers to Part A and B.

C1. Define a predicate hits(B,P) that holds if and only if a player whom is a beater, B, hits another player, P, (with their club) - here assuming that they automatically hit any player who comes within a 1 metre radius of their position.

C2. Define a predicate end_of_game(T1,T2) for a game between teams T1 and T2 that holds if and only if the snitch (ball) has been caught by the correct player from one of the teams whilst no player from either team was committing a foul and the formations of both teams are valid..

C3. Define a predicate update_score(T1, S1, T2, S2) that updates the scores (S1 and/or S2) based on either a goal being scored by either team or when the snitch (ball) has been caught by the correct player from one of the teams. In quidditch - "*Each goal scored is worth ten points. .... Capturing the Snitch earns the Seeker's team 150 points"*

Once completed, uncomment the Part C loading part in your base file (*base.pl*) so that these predicates are automatically loaded from *base.pl.*

Again in order to test part C you will need to define a number of game configurations using player/4 and ball/4 (for player and ball positions). Add these test examples (and results) into your separate prolog file *game.pl* such that you can comment in and out different game configurations for testing. Clearly comment the tests that apply to Part C.

Hints and Guidance - It is assumed you are familiar with the concepts introduced in lectures, in the lab sessions and those within the "Extra Topic 10 – Working with Files".

Marking Scheme

The marks for this practical will be awarded as follows:

- **Part A –** Prolog code predicates as outlined 20%
- **Part B –** Prolog code predicates as outlined 30%
- **Part C** – Prolog code predicates as outlined 20%
- Testing – a provided set of relevant test examples (game.pl) that show the correct operation of your code for parts A-C 20%

- Clear, well documented and structured program source code (for all parts) 10%

Total 100%

Submission

You must submit the following:

- Full program **source code** for your solution to the above tasks in separate files for A – C (named as outlined above – partA.pl, partB.pl, partC.pl and game.pl).

  Do not submit any other files – any other files submitted will be ignored. Your code will be tested using the standard base.pl file issued with the assignment in addition to a range of unseen testing examples to emulate game play using variations in player and ball positions with varying game scores (via a game.pl file containing varying player/4, ball/4 and score/2 predicates).

  Make sure you program source code is fully commented (as per the provided examples).

Plagiarism : You must not plagiarise your work. *You may use program source code from the provided course examples or any other source BUT this usage must be acknowledged in the comments of your submitted file. Automated software tools may be used to detect cases of source code plagiarism in this practical exercise, accounting for provided exemplar code, which may include automatic comparison against code from previous year groups. Attempts to hide plagiarism by simply changing comments/variable names will be easily detected.*

You should have been made aware of the Durham University policy on plagiarism. Anyone unclear on this must consult the module lecturer prior to submission of this practical.

Submission

To submit your work create a directory named by your DUO login username (e.g. hplq98). Place all required files in this directory. **ZIP** (not rar or .z7 etc.) this entire directory structure and submit it via DUO *(late submissions will be penalised following school policy).*

Submission Deadline : 2pm (UK time) – 2nd March 2015