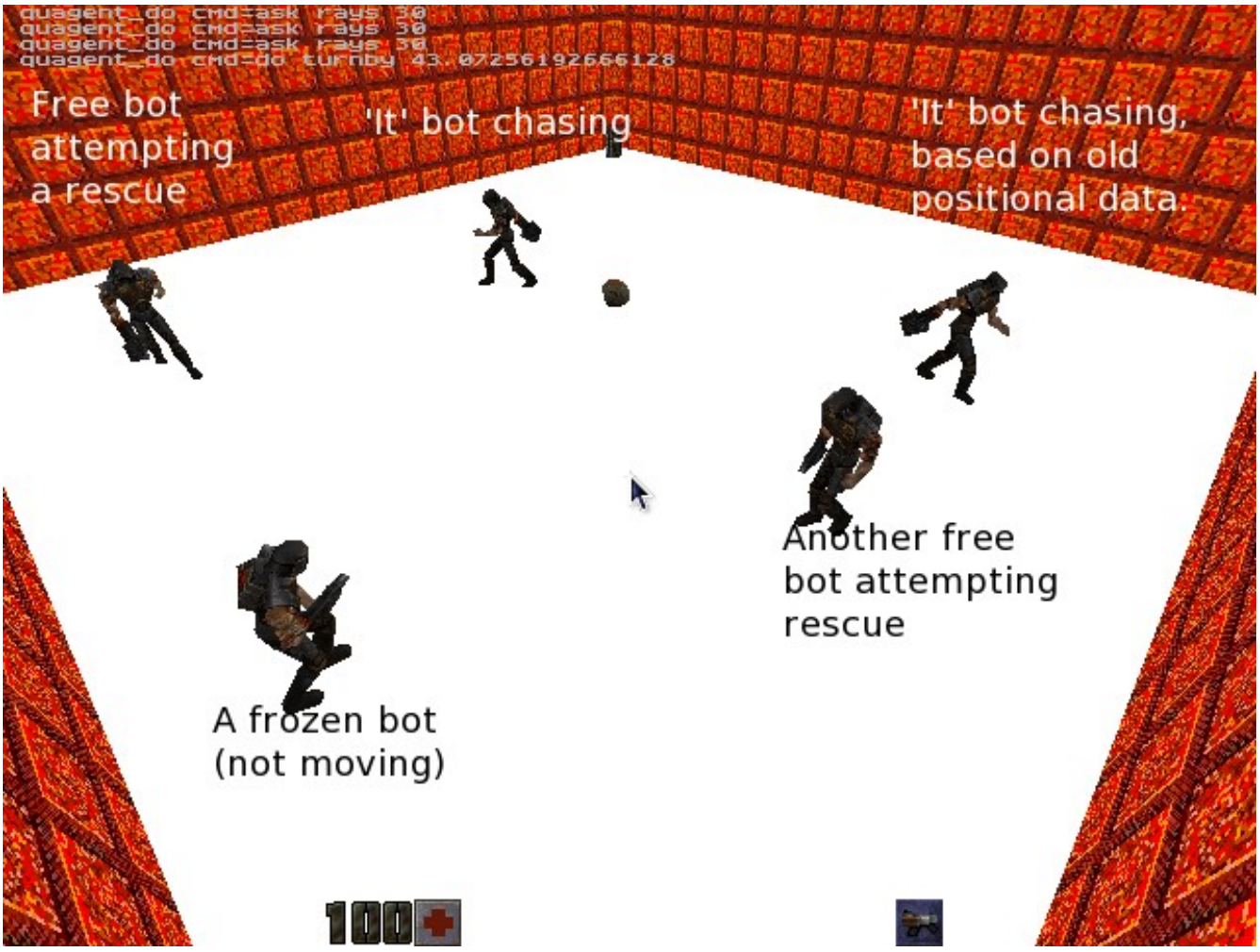
Introduction

Simply put, these quagents play that staple childhood game, freeze tag. The rules of freeze tag are simple: a rectangular field is set up, and some subset of the players are marked as "it". The two basic actions of the game are tagging and moving: obviously tagging someone implies some positional proximity, so in general free players flee the it players, while the it players chase the free. When a free player is tagged by an it player, he becomes frozen, and cannot move or tag. When a frozen player is tagged by a free player (a "rescue"), she becomes free, and may move and tag once again. The goal of the it players is to tag all free players. The goal of the free players is to not let this happen (often, the game runs until everyone is too tried to keep playing, or has no distinct ending; there is no victory condition for the free players).



Behaviours

Precisely what constitutes a distinct behaviour is difficult to define in this game: there are effectively only three actions that agents can take: tagging, walking, and turning. Because turning and walking are a little too coarse to capture the actions of a agent, we will consider behaviours as those things that drive an agent's decisions: that is, "where do I walk". In the game of tag, standing still is always a losing strategy, so under this implementation agents always walk as far as possible, and simply adjust their direction of motion constantly to find a destination. The factors that determine their movement are as follows.

It's worth noting that all behaviour is entirely greedy, and yet the agents seem to operate as if they have a plan. This illusion is caused by the fact that if moving in a certain direction is ever favourable, after moving slightly in that direction it is usually still favourable. In a way, the freeze-tag space can be thought of as a flow field, which responds to the positions of the people within it, who respond to the flow field.

Perception

Since an agent can be treated differently depending on whether he is free or frozen, it is necessary to have a way to tell agents apart. Each agent must then announce his position - which requires agent-to-agent communication. This has some interesting implications on the program structure, which is discussed later. However, even if an agent was assigned a unique identifier and allowed to announce its status, other agents have no method by which to determine who that agent is in order to establish his position.

So, in a way, the agents are playing a weird hybrid of freeze tag and "marco-polo". Instead of tracking other agents, each agent is honour-bound to truthfully announce his location and status when the referee whistles. Agents still do use some active perception, however: to determine the size of the space that they are playing in, they constantly look around for the walls.

Flight

Flight is potentially the most obvious behaviour in a game of tag. In short, agents avoid agents who are "it". The general process is as follows: the agent first considers all the known locations of players (some of which may be up to 100 milliseconds old - this causes some unusual behaviour, which can be noticed in the above screenshot). Next, the agent weighs its desires (desire to rescue, pursue, flee, guard, and avoid bunching up), and compares them to the positions of the other agents. The closer a free agent is to an it agent, the stronger their desire to run in the direction opposite that closest to the agent. As an example, a single agent in a

Agents can be run with a variable flight coefficient (which determines how important this behaviour is in determining general motion). Agents with a remarkably high flight coefficient tend to become trapped by pursuers in corners, since they do not have a relatively high priority on avoiding walls. Agents with a remarkably low prioritization of flight tend to (predictably) fail to flee when they probably should. The strength of a flight impulse from an 'it' agent is inversely proportional to the distance to that agent. The overall direction of flight is the weighted average of these impulses.

Pursuit

Pursuit is the natural foil to flight: while free agents flee it agents, it agents seek free agents. This behavioural parameter can be thought of as flight with a negative (attractive) coefficient: agents prioritize chasing close agents over distant agents, and react to the sum of agent positions (so two agents twice as far to the left as one agent is to the right have an equal "I-should-chase-that-way" score: $2x/2d = x/d$).

It players with a high pursuit coefficient tend to do well in games where they have a relatively high population relative to the free players (at least half as many players are 'it' as are 'free'), or in games in narrow spaces. They often are outmanoeuvred in games in large spaces or with a high ratio of free players to chasing players, since they bunch up and do not guard tagged players as well.

Rescue

Free agents can have a desire to rescue frozen players as well. This behaviour works much like pursuit: the free agent attempts to minimize distance from frozen players, prioritizing closer frozen players over more distant ones, and clumps over solitary standing players. With high rescue to flight coefficient ratios, this can cause a free quagent to get into a "tag-war" with an 'it' agent over a frozen agent: the free agent will stand on the opposite side of the frozen quagent and repeatedly tag him, while the it quagent will continuously tag the suddenly free agent.

Agents with high rescue priorities relative to flight tend to perform poorly (five trials reveals that the team survives on average for half the time), and agents with low rescue priorities relative to flight tend to also perform poorly, since the tagging team soon outnumbers them. Agents with an approximately equal rescue-to-flight ratio tend to survive longest in adverse conditions.

Spacing and Wall Avoidance

Agents with a positive spacing coefficient don't like to bunch up, and they dislike being next to walls (and especially in corners). For the purpose of avoiding bunching up, agents effectively flee from other agents of the same type (so, it agents spread out for a "zone defence" and free agents spread out to make it more difficult to catch them). A low spacing coefficient is extremely problematic, since free agents are very easily driven into corners by it agents, and it agents often get caught up in collective pursuit of a single free agent (enabling other free agents to go and rescue frozen agents).

Wall avoidance is a little more complicated. Because the space is rectangular, every agent periodically sends out vision rays, establishing the positions of the walls - the x and y coordinates of these are taken as the boundaries of the field, and all walls exert a repulsion influence of magnitude c/distance on agents with a spacing coefficient of c. This repulsion is always perpendicular to the wall, so the net effect is stronger and oblique to walls near the corners.

Since rays can be blocked by other agents, the agent assumes that the field never shrinks, and in the case that it gets a different coordinate for the "wall" on one call instead of another, it assumes the larger space is more representative. Of course, one can see that this poses problems for non-rectangular spaces, or spaces that have an interior obstacle structure.

Agents with high spacing coefficients tend to gravitate toward the centre, at least until other agents do, in which case they establish equal spacing around the room. The flexibility of an agent to move into different sectors of the room is determined by the spacing coefficient.

Guard

Guard behaviour is the 'it' analogue to rescue behaviour: agents try to minimize distance to frozen agents. Because it is not necessarily a good idea to go closer to the closest frozen to guard, guarding is implemented with a shift and different scale than all other behaviours, which makes the coefficient difficult to estimate optimally.

In general, taggers with low or zero guard coefficient do not perform substantially worse than those with high coefficients, and a high ratio of pursuit to guard coefficients means that a tagger will never leave a tagged free agent. Since the 'it' agents generally tend to be close to the frozen agents (they did tag them, after all), and all agents move at the same speed, the additional incentive to guard is typically not necessary. If agents moved at different speeds and the field was much larger, however, it is anticipated that guard behaviour becomes more important.

Tagging

Finally, the action of tagging does not influence movement but is instead a result of it. One agent tags another when it is stopped short of its movement goal and it knows there is an agent nearby that needs tagging (if the stopped agent is it, an "agent that needs tagging" is defined as a free agent, while if the stopped agent is free, this refers to frozen agents). Since agents stop immediately once the referee (TagCommander) tells them that they have been tagged, this can lead to problems like a tagging quagent trapping himself within frozen quagents who are no longer allowed to move: a so-called "freeze quag-mire".

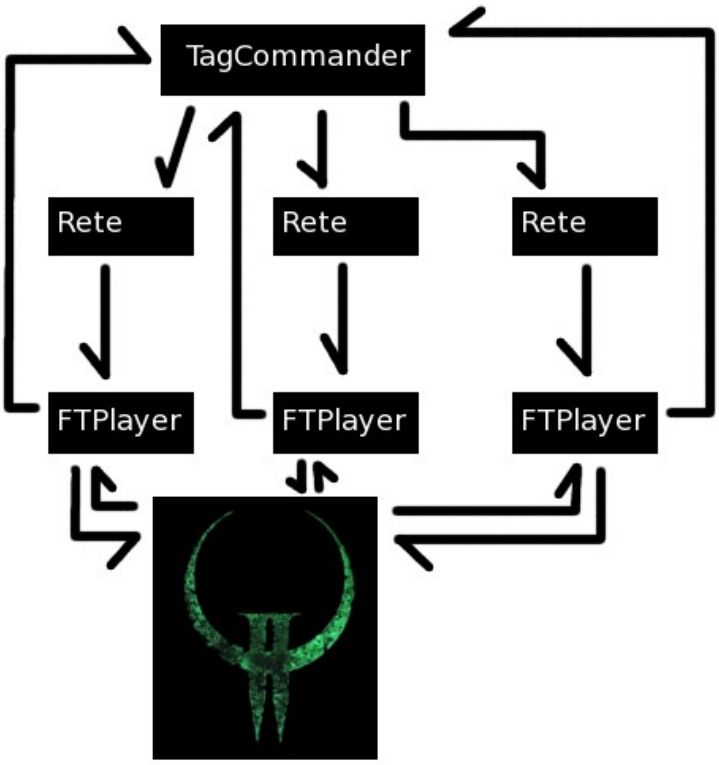
(The above agents have equal flight, pursuit, and wall-avoidance coefficients - eventually (usually ten minutes or so), the pursuing agent will catch the fleeing one, since he walks a very slightly smaller circle. The grenades are placed at the centre of the room, and the agents orbit around them.)

Program Structure

To facilitate bot-to-bot communication (the need for which, at least using quagents, is discussed above), and to allow for a variable number of bots with different knowledge bases, a somewhat convoluted structure that moves a great deal of control-structure-esque code out of Jess and into a higher level referee program is necessary. A referee program called "TagCommander" is responsible for creating N free tag players and M it tag players (N and M are taken as command-line arguments) and synchronizing their announcements to ensure a fair game. The .clp file is not run directly, but is instead controlled through "(run)" commands sent from the TagCommander, and many of the functions within it are called from either it's subsidiary Java code or the Commander.

The general problem is that while Jess-to-Java communication is easy, it is difficult to pass a reference to the active Jess program from Jess to Java. As a result, communication is initially established through a static-reliant loop: TagCommander contains a static variable that refers to itself, which the lower-level Java implementations of an agent can refer to. They then ask the TagCommander for a reference to the Jess Rete object that contains the knowledge base, and can then run commands in it asynchronously (that is, Jess does not have to poll Java, asking "did I find out that I've stopped yet?" - instead Java can simply call logic.executeCommand("(assert (stopped))")).

The general flow of communication: because TagCommander has references to the Rete objects, the FreezeTagPlayer objects can easily get references to their logical Rete objects.



Appendix A: Jess Code

The following is precisely the "batch" Jess code that is loaded into the tag players when the program starts. As explained in the Program Structure section, this Jess code is rather anaemic, since that majority of information flow is *into* Jess from TagCommander and FreezeTagPlayer: in other words, Java is responsible for asserting things, maintaining the state of variables, and calling (among the other functions), "(run)" and so on. Do to text flow alignment, some of the below comments run multiple lines when in reality they run only one.

```
; This file is the Jess file for the FreezeTagPlayer. It should not be
run directly
; but is instead loaded through the TagCommander class.
; FreezeTagPlayer must respond to an (i-am-it) fact that sets it up for
it-dom.
; FreezeTagPlayer must NOT have a (run) call. (run) is called from
TagCommander, as is (initialize).
; Joshua Pawlicki, Feb 25
(import quagents.agents.FreezeTagPlayer)

(defglobal
  ?*frozen* = FALSE
  ?*agent* = 0
  ?*whoami* = 0
  ?*state* = (assert (free))
)

(deffunction initialize ()
  (bind ?*agent* (new FreezeTagPlayer ?*whoami*))
  (?*agent* initialize)
  (?*agent* getWhere)
)

(deffunction addPlayer (?x ?s)
  (?*agent* addPlayer ?x ?s)
)

(deffunction hearShout (?i ?s ?x ?y)
  (?*agent* hearShout ?i ?s ?x ?y)
)

(defrule am-it
  (i-am-it)
  =>
  (retract ?*state*)
  (bind ?*state* (assert (it)))
  (?*agent* setState 2)
)

(deffunction tick ()
  (?*agent* getWhere)
  (?*agent* getWalls)
  (?*agent* broadcastPosition)
)

(deffunction tick2 ()
```

```
  (?*agent* updateHeading)
)

(deffunction tagged (?x)
  (if ?x
    then (i-am-frozen)
    else (i-am-freed)
  )
)

(deffunction i-am-frozen ()
  (bind ?*frozen* TRUE)
  (retract ?*state*)
  (bind ?*state* (assert (frozen)))
  (?*agent* amFrozen)
)

(deffunction i-am-freed ()
  (bind ?*frozen* FALSE)
  (retract ?*state*)
  (bind ?*state* (assert (free)))
  (?*agent* amFree)
)
```

Additional Java code, necessary for the understanding of FreezeTagPlayer, has also been submitted with this report.