CS 312 Problem Set 6: λ Ball–Specification Change

Assigned: April 10, 2003 Due: 11:59PM, May 2, 2003
Spec Change: April 20, 2003

1 Introduction

As stated in the original writeup of the problem set, this project is now to have specification change. If your code is designed modularly and robustly, the changes necessary should be minimal. However, if your code thus far does not take well to change, some of these changes could take extra time to implement.

2 Powerups

The change in the specification affects spawning and destroying robots. Until now, bots have had the ability to spawn an infinite number of bots. The only disadvantage to doing so is that every one of the bots on the team slows down when performing actions. The board now includes two powerups: *spawn* and *overload.* These powerups appear on the board with a certain probability each clock cycle. The spawn powerups appear with probability $\frac{1}{1024}$ and the overload powerups appear with probability $\frac{1}{4096}$. This means that a spawn powerup will appear *on average* every 1024 clock cycles.

At any given time, only eight powerups should be on the board. A powerup appears on a free tile, which is a tile with no bot, ball, or other powerup on it that is not in the goalie area or goal area. When a bot enters a tile containing a powerup, it immediately picks up that powerup, which then disappears from the board. If the ball tries to enter a square with a powerup, it should bounce back in the direction from which it came.

2.1 What the powerups do

The spawn powerup gives the team the ability to spawn a new teammate. The team must collect three spawn powerups before being able to spawn a new bot. Collecting a spawn powerup gives the team one additional spawn credit. *This means that teams no longer have unlimited spawning ability.*

Only the goalie bot can spawn a new bot. When a bot is spawned, it appears at a randomly-chosen empty tile immediately in front of the team's goalie area. Additionally, the number of spawn credits decreases by 3. If a team did not have at least 3 spawn credits, then they cannot spawn a new teammate. It is important to note that a bot does not necessarily know if the team has enough spawn credits and therefore might try to spawn a bot when the team should not be able to. The world should catch this fact and immediately terminate the new process. The world should also enforce the condition that only goalies for the teams are allowed to spawn bots. The bot that attempted to spawn illegally is returned ~1 as the result of the spawn and is allowed to proceed normally. A team initially starts with 9 spawn credits, i.e., the ability to create 3 bots.

When a bot picks up the overload powerup, it gets the ability, for 3000 clock cycles, to destroy any bots it runs into or bots that run into it. After that time, the bot itself self-destructs and explodes, causing splash damage. The splash damage kills any non-goalie bot from either team within a one tile radius. When a bot is destroyed or self-destructs, its process is killed off by the world and the graphic for the bot is removed from the board.

A bot that has picked up an overload powerup and tries to move into a tile with another bot moves into that spot always wins the fight and destroys the bot in it. If a bot tries to move into a tile with a bot currently overloading, the moving bot is destroyed instantly. If an overloaded bot tries to move into a spot with another overloaded bot, then both the overloaded bots are destroyed causing splash damage in the one tile radius.

If an overloading bot has the ball when it self-destructs, then the ball is left stationary on the tile where the bot exploded.

Overloaded bots are represented as T_OVERLOADEDOPPONENT or T_OVERLOADEDTEAMMATE as appropriate in the list returned by LOCABS or in the status list returned after any action. If an overloading bot has the ball, then it is still represented by one of these two constants.

2.2 Changes to problem set

A bot must be able to know when it is overloading. Therefore, the status list returned by any action now has an additional element. The status list now has the format $[ballradius, ballmajdir, ballmindir, adj, coord, ovl]$:

ballradius	the radius at which the ball is relative to the bot
ballmajdir and *ballmindir*	are the best two directions in which to move (one after the other) that will lead the bot closest to the ball from the current position, assuming the ball does not move. If the bot has the ball, both values are 0. If the ball is in a straight line from the bot, both values will be the same–the direction to the ball.
adj	is a list of the items in the six adjacent tiles, indexed by the direction.
coord	is the coordinates of the current bot
ovl	is 1 if the bot is overloading, 0 otherwise

Additionally, there are new constants defined to refer to these powerups in the adjacency list, T_PSPAWN and T_POVERLOAD.

The specification for the A_HEADCOUNT changes as well so that a bot can determine how many spawn credits are left:

Command	**Base Time**	**Args**	**Description**	**ASR**	
A_HEADCOUNT	AT_HEADCOUNT	None	Determines the PIDs of the teammates and opponents	$(s, (l_1, l_2))$	where s is the number of spawn credits left for the team, l_1 is the list of PIDs of the teammates, and l_2 is a list of opponents' PIDs, both ordered by PIDs.

2.3 Graphics

The graphics system has been augmented to include graphics for the overload powerup, spawn powerup, overloading bots, and an exploding bot. All graphics are still in the gfx directory. In addition to the graphics described in the original problem set, the following images are now also available:

File	**Description**
blue*i*o	An overloading blue robot looking in direction i
red*i*o	An overloading red robot looking in direction i
blue*i*bo	An overloading blue robot looking in direction i with the ball
red*i*bo	An overloading red robot looking in direction i with the ball

For examples of the new graphics, see Figure 1.

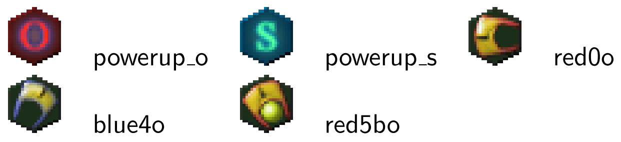
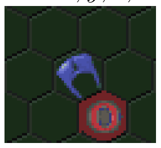


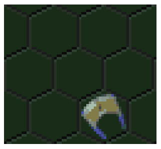
Figure 1: Examples of new graphics in the game

2.4 Examples of actions

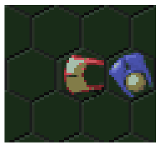
Below are some examples of the actions and what they'd return, given the existence of powerups now. The bot is assumed to be at position $(0, 0, 0)$ when starting. Any values that can't be determined are given by variables $x, y, z, \ldots$.



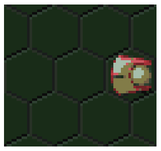
do T_MOVE $\Longrightarrow$



```
([x,y,z
[T_EMPTY,T_EMPTY,T_EMPTY,T_EMPTY,T_EMPTY,T_EMPTY],
[0,~1,1],1],R_OK)
```



do T_MOVE $\Longrightarrow$



```
([0,0,0
[T_EMPTY,T_EMPTY,T_EMPTY,T_EMPTY,T_EMPTY,T_EMPTY],
[~1,1,0],1],R_OK)
```

3 Tasks

The tasks for the problem set are the same as they were before the specification change. However, the world you turn in should be one that handles the powerups. You do not need to submit a separate version of the world that conforms to the specifications before the change. Note also that the bots created for this assignment will be tested in a world with the powerups.