ClobberBot

or

How I Learned to Stop Worrying and Love the Agents

Quincy Bowers

Boise State University

Fall 2011

Introduction

In this project I designed an agent for the ClobberBot game. The game consists of an arena where bots, simple reflex agents, battle each other by maneuvering and firing bullets at each other. The game is played in discrete time steps. At each time step the bots in the game each make a decision to take a single action. They can either move or shoot in one of eight directions. A bot can also choose to do nothing. The game server applies each of the bot's actions, updating the game state.

Design

Dodging

I designed my bot by starting with the purely random bot code provided. My strategy was to work on dodging first since being alive at the end of the game is worth a lot more points than kills. My bot looks at each bullet in the game and determines its trajectory. It then calculates whether that trajectory intersects with the rectangle that surrounds it and how far away the bullet is. If the bullet will intersect and the bullet is within 28 pixels the bullet is considered dangerous. I chose 28 pixels through testing. Originally, I picked 16 pixels which would mean the bullet would hit the bot in 4 time steps. In 4 time steps, my bot can move 8 pixels which should be more than enough to get out of the way. But testing showed an improvement as I increased the size of the "danger zone", which leveled off around 28 pixels. The closest dangerous bullet is the only bullet that the bot reacts to.

If a dangerous bullet is found the bot reacts by attempting to move out of the way. The direction to move is calculated by the `getDodgeDirection` method by drawing an imaginary line through the bot which is parallel to the bullet's trajectory. I then calculate which side of the line the bullet is on in order to decide which direction leads to a quicker escape.

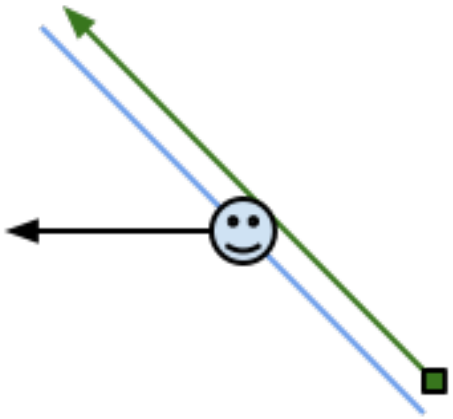

Illustration 1: The bot will move to the left to escape the bullet.

For vertical and horizontal bullet trajectories it is simple to calculate which side of the line the bullet is on. Diagonal lines are more difficult though. We can use the end points of the line, and the location of the bullet, to calculate a cross product which will tell us which side of the line the bullet is on.

$L = x_{1,} y_{1,} x_{2,} y_2$ The line drawn through the bot.

$B = x_b, y_b$ The bullet position.

$[(x_2 - x_1) \cdot (y_b - y_1) - (y_2 - y_1) \cdot (x_b - x_1)] < 0$ when the bullet is on the "left-hand-side" of the line.

Attacking

In order to attack, the bot determines which bot is the closest to it and designates it as the target. The bot ignores any bot it finds which has an ID matching one of the ID's in its list of team mates. If a target is found and the bot can fire, it calculates the cardinal direction that most closely matches the actual direction to the target and fires in that direction. This calculation is performed with the following equation.

$$\theta = atan2(\Delta x, \Delta y)$$

This calculates the angle from a horizontal line passing through $(x_{1,} y_1)$ to the line described by $(x_{1,} y_{1,} x_{2,} y_2)$. Then this angle is subtracted from each of the angles representing the cardinal directions and whichever one yields the smallest difference is the direction in which the bot fires.

However, if there is a target but it is in between shot frequencies the bot moves toward the target in the same direction it would have fired. But it only moves toward the bot if the current distance to the bot is greater than 100 pixels. This is to try to maintain a safe distance from enemies.

Avoiding Walls

At this point the bot worked fairly well but it was easily trapped against walls and in corners where it was summarily executed. To mitigate this, there is a 50 pixel buffer along each wall and if the bot finds itself with no other action to take while inside this buffer it retreats from the wall. This helped quite a bit in keeping the bot alive longer.

If no other action is selected the bot will do nothing at all. I tried doing random movements but these seemed to reduce the bots effectiveness. Occasionally, though, a situation is encountered where the remaining bots on the screen will get setup just perfectly so that they all remain still and shoot at nothing until the end of the game. In order to try to get my bot to do something more intelligent I created an idle counter. Any time my bot decides to sit still, the idle counter is incremented. When the bot makes a move as described above the idle counter is

reset. If the idle counter ever becomes larger than 20 the bot makes a random move. Testing the effectiveness of this change is difficult, but it doesn't seem to decrease effectiveness.

Testing

To facilitate testing I wrote a short Perl script that runs the Clobber program a specified number of times and prints a summary of the scores for each bot across all of the games played. This script is called `test.pl` and is included in the submission.

qbowers vs Random Bots

I ran 100 games against the random bots to show that my bot easily dominates over purely random agents.

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	302	87	19699	13
ClobberBot4 by Tim Andersen	31	3	2045	97
ClobberBot5 by Tim Andersen	29	2	1841	98
ClobberBot2 by Tim Andersen	17	2	1048	98
ClobberBot3 by Tim Andersen	23	4	1009	96

1vs1 Intelligent Bots

Next I ran 100 games against each of the more intelligent bots provided as 1-on-1 matches. The results in the following table show that my bot is pretty effective in 1-on-1 matches. The p3 bot is the only one that is not dominated by my bot. The results for the games against p3 show that these games end in a draw more often than anything else.

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	85	99	9784	1
p1	1	15	772	85

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	63	65	6496	35
p2	35	37	4012	63

PLAYER	KILLS	WINS	POINTS	DEATHS
p3	3	99	5547	1
qbowers	1	97	5337	3

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	68	99	8648	1
p4	1	32	1976	68

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	100	99	10417	1

p5	1	0	79	100

PLAYER	KILLS	WINS	POINTS	DEATHS
p6	85	90	9163	10
qbowers	10	15	1357	85

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	79	98	9377	2
p7	2	21	1199	79

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	76	81	8242	19
p8	19	24	2278	76

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	97	97	10126	3
p9	3	3	374	97

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	86	88	9054	12
p10	12	14	1454	86

PLAYER	KILLS	WINS	POINTS	DEATHS
qbowers	53	74	6646	26
p12 bot by Jeff Cope	26	47	3938	53

qbowers vs Intelligent Bots

Next I ran tests with all of the intelligent bots at once. I placed more importance on this series of tests so I ran 1000 games instead of just 100. The results in Table 3 show that my bot remains pretty effective even in large free-for-all games.

PLAYER	KILLS	WINS	POINTS	DEATHS
p6	1944	475	1220508	525
p12 bot by Jeff Cope	1856	132	998079	868
p8	1803	167	974157	833
qbowers	1685	197	942173	803
p10	1068	63	570567	937
p2	862	4	416626	996
p3	540	21	282547	979
p9	387	3	199319	997
p4	253	11	134724	989
p1	214	5	109315	995
p5	202	1	105152	999
p7	194	4	101657	996

Teams

Finally I tested my bot's effectiveness in team settings. I ran 100 games each with team sizes of 2, 3, and 4. There seems to be an increase in effectiveness as the team size increases. My bot moves from 4th to 3rd place going from a team size of 2 to a team size of 3. This increase could be due to the sheet number of available targets but you would expect to see a similar increase from the other bots if that was the case.

Team Size 2

PLAYER	KILLS	WINS	POINTS	DEATHS
p6	408	64	48930	136
p8	386	13	40152	187
p12 bot by Jeff Cope	387	11	38893	189
qbowers	362	23	37772	177
p10	200	5	20848	195
p2	168	0	16830	200
p3	114	2	11795	198
p9	112	1	11126	199
p1	49	0	5527	200
p4	50	0	5149	200
p7	42	0	4138	200
p5	33	0	3148	200

Team Size 3

PLAYER	KILLS	WINS	POINTS	DEATHS
p6	555	73	97838	227
p8	581	14	88987	286
qbowers	545	33	86824	267
p12 bot by Jeff Cope	558	4	80880	296
p10	325	2	48665	298
p2	316	0	48260	300
p3	192	3	26889	297
p9	175	0	25655	300
p4	85	0	14227	300
p1	76	0	12870	300
p5	57	0	9239	300
p7	51	0	8753	300

Team Size 4

PLAYER	KILLS	WINS	POINTS	DEATHS
p6	725	81	167756	319
qbowers	702	41	150036	359
p12 bot by Jeff Cope	739	5	149820	395

p8	737	11	141093	389
p10	441	2	87695	398
p2	410	0	80280	400
p3	320	1	64074	399
p9	277	0	57373	400
p4	101	0	22674	400
p1	117	1	22236	399
p5	86	0	18417	400
p7	65	0	13239	400

Conclusion

Implementing a reflex agent to play the ClobberBot game was fun, challenging, and instructive. I am pleased with how my bot turned out. It plays well but there is room for improvement. My bot only reacts to the nearest bullet or bot. This means it can easily get trapped in situations that might be otherwise be escaped if it took into account more of the environment.