Description

The project will develop a program to simulate one or more (depending on time) robots playing the 2016 FIRST Robotics Competition (FRC) game, Stronghold.

Overview

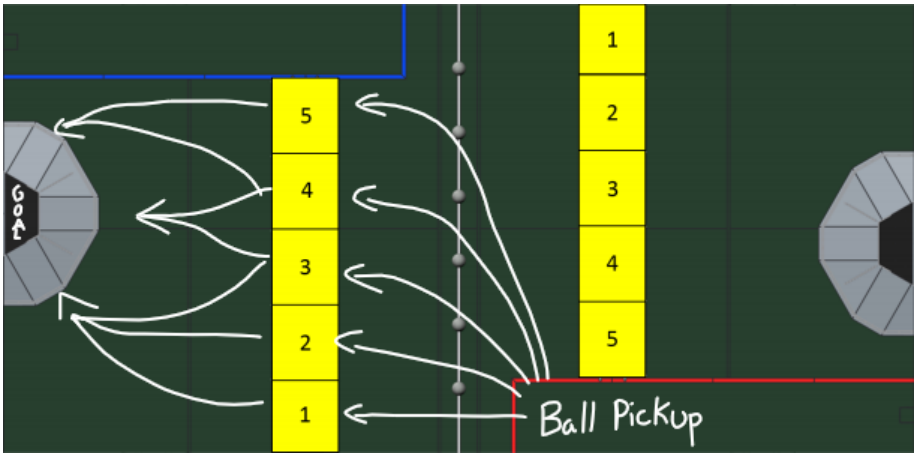

Figure 1: A map of the field with defense positions enumerated, pickup/goal marked and arrows indicating robot traversal.

Stronghold is the game played for the 2016 FRC season. In it, robots pick up ball from one end of the field, cross the defenses, and score by either launching it into the high goal or rolling it into the low goal. There are 4 pairs of defenses, and one of each is selected for the match, with a low bar being used in position 1. Points are awarded for scoring a ball and the first 2 crossings of a defense. Given the number of variables at play, both from field and robot variations, it can be difficult to determine the optimal strategy. This project will develop a program which takes a variety of information about a robot and the field configuration and determines the optimal cycle pattern based on information such as defense cross times and scoring areas.

Resources

The code is written in C++ and was developed using Visual Studio Code with C/C++ extensions and a custom buildscript (makefiles are generated for portability). Details pertaining to Stronghold such as locations of field elements and point values will be taken from the official game manual. The application is console-based but features a graphical representation of the graph using SDL2.

File Formats

robots.csv

The primary configuration file is robots.csv. This file contains the various robot information used while running the simulation in the comma-separated value format. This format was chosen as it is easy to parse and allows for easily adding multiple robots if time allows. Values for shot parameters are limited based on physical constraints of the real field (you can't hit an opening from an overly shallow angle). Unsigned chars are used frequently as most values do not need to exceed 255.

Table 1: Data stored in robots.csv

Name	Type	Description	Values
team	int	Team number, used in output	0-9999
can_low	boolean	Flag indicating if robot can go under low bar	0 or 1
shot_range	unsigned char	Maximum range for high goal shot [inches]	0-170
centre_shot_time	unsigned char	Time needed for centre high goal shot, 0 indicates unable [seconds]	0-255
side_shot_time	int	Time needed for side high goal shot, 0 indicates unable [seconds]	0-255
centre_angle	unsigned char	Maximum angle for shot at centre goal [degrees]	0-40
side_angle	unsigned char	Angle for shot at side goal [degrees]	0-40
low_time	unsigned char	Time needed to store in low goal 0 indicates unable [seconds]	0-255
defenses	long int (hex)	Hex value representing time to cross defenses , 0 means cannot cross and F means it takes 15s [seconds]	See Table 2
speed	unsigned char	Robot's speed [fpm]	1-255
point_value	float	How many seconds of weight are removed per point scored, higher numbers mean more defense crossing	0-9999

Shot Locations

Shot locations are calculated as shown in Figure 2. All shot locations are placed along the circle with radius R, as the most efficient shot will always be the longest. The points for the side goals will be placed at angle θ from the wall. The points for the centre goal will be placed at ±α from centre, and directly out from centre. These positions have been chosen as the allow for maximum flexibility in robot capabilities without major impact on performance.

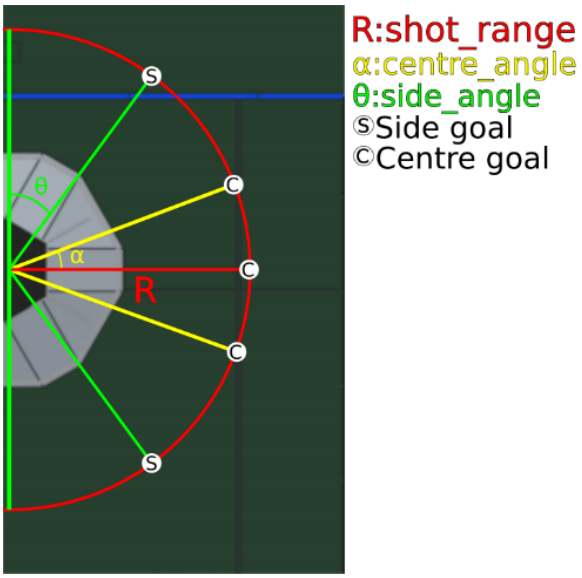

Figure 2: Illustration demonstrating calculation of shot locations.

Defense Values

Table 2: Table listing the positions in the bitmask for the 8 variable defenses.

Category	Defense	Position (Denoted by **F**)	ID	Shift
A	Portcullis	0x0000000**F**	0	0
	Cheval de Frise	0x000000**F**0	1	4
B	Moat	0x00000**F**00	2	8
	Ramparts	0x0000**F**000	3	12
C	Drawbridge	0x000**F**0000	4	16
	Sally Port	0x00**F**00000	5	20
D	Rock Wall	0x0**F**000000	6	24
	Rough Terrain	0x**F**0000000	7	28

Ex. A team that can cross all the drive-over defenses in 3 seconds would have: 0x00330033
A team that can do that and cross the Portcullis in 8 seconds would have: 0x80330033

Program Operation

Configuration + Setup

The program will load robot information from ./robots.csv or a specified file following the format in File Formats, and take user input for the field configuration and simulation duration. In the event of invalid configuration, errors will be thrown describing the issues. Once configuration has been loaded and validated, the robot-dependant shot nodes will be calculated and added to the graph.

Simulation

Once setup is complete, the simulation will be run. This will consist of navigating the robot through the generated graph until the predetermined time limit has been reached. Navigation is based on Dijkstra's algorithm, with path weights being time-based, and weight removed for points scored. The target node is either the goal node, a node at the centre of the tower with 0-weight edges to the various shot locations, or the node at the end of the passage where balls enter the field, depending on whether the robot has a ball. The robot is then moved to the new location and is set to "sleep" until a set time. When the simulation hits that time, it will generate the event with information about location, time, and planned path, before the robot is moved again. Each event (moving to another node, crossing a defense, scoring, etc.) will be added to a queue for later playback.

Output

Upon completion of the simulation, the program will list the events from the queue sequentially, displaying the result to the user. The user will be able to step through the event list, getting the total score and defense values at the time of the selected event. The field GUI will also show the robot's position, as well as a connection to its previous position and the planned path.

Error Messages

There are a few error messages that may occur when preparing the simulation.

File not Found

This error will be thrown if ./robots.csv or specified file cannot be found. The output will state the exact path used.

Parser Error

This will be thrown if an error is encountered while parsing robots.csv, most likely caused by missing or invalid inputs. The output will state the number of items read successfully to assist in narrowing the possible causes.

Duplicate Defense Category

This will be thrown if two defenses from the same category are selected for use on the field. The output will inform the error

Shot Computation Warning

This will be thrown if the values used when computing the shot locations are outside the ranges specified in robots.csv. The output will state which value(s) are problematic. However, unlike the above errors this one will give the option to proceed as the erroneous data will only impact the usefulness of results not the success of the simulation.

Overall Design

Configuration

```
Load from provided configuration file
 - fgets + sscanf
Handle/report any errors in loading
Get defense configuration from console
 - Defense IDs space separated in order from Figure 1 (position 1 low bar automatic)
 - If 2 defenses from same category chosen, then warn with override
```

Setup

```
Build graph
 - Hardcoded positions from game manual
 - Include nodes for tower, defenses, courtyard, and centre-field
 - Add goal node at centre of tower with 0-weight edges to scoring locations
Calculate shot locations (see Shot Locations)
Add each to graph
 - Connect to predetermined nearby nodes
 - Connect to goal node
Set intakeNode to end of passage
```

Simulation

```
while time < duration
  on robot with lowest wake time
    log event for current location
     - if location is connected to goalNode
       - add points(high or low)
       - remove ball
     - if location is defense
       - add points
       - reduce defense health
     - if location is intakeNode
       - get ball
    if robot has ball
      find least-weight path to goalNode
       - Add weight for time taken
       - Deduct weight for points crossing a defense
       - Deduct weight for points from scoring
      advance to next location on path
      set wake time to time until new location
    else
```

```
            find least-weight path to intakeNode
             - Add weight for time taken
             - Deduct weight for points crossing a defense
             - Deduct weight for points from scoring
            advance to next location on path
            set wake time to time until new location
          set time to lowest waketime
```

Output

```
      while not escape
        if down
          index ++
              - prevent overflow
        if up
          index --
              - prevent underflow
        update console
         - write header
         - write events around selected
           - calculate score and defense stats
         - write scoreboard and defense stats
        update GUI
         - draw graph (if visible)
         - draw robot
         - draw path lines
```