Abstract— **This document describes and explains with elaboration Team NYCasuals's strategy in participating in the RoboCupJunior Rescue Simulation (former CoSpace) challenge. All present and future works are discussed in this document.**

I. INTRODUCTION

Team NYCasuals is a robotics team from Nanyang Robotics Club (NYRC). Our club believes that passion, dedication and resilience are the drivers to success. Our team is passionate about robotics and enjoys the process of self-directed learning while experimenting with the CoSpace platform. In 2016, our efforts paid off and we claimed the world champion title in RoboCup 2016 held in Leipzig, Germany. Additionally, we successfully defended our title at RoboCup 2017 in Nagoya, Japan. At the competition, we met many talented programmers whom we learnt many unique ideas, strategies and techniques from. Our goal in 2018 is straightforward: we shall defend our world champion title. At the same time, we are looking forward to the creative strategies that the other national champions employ, and learn from them.

II. PROGRAMMING LANGUAGES USED

1) C
2) C++
3) Python

III. PROGRAMMING APPROACH

Via teamwork, collaboration and strategical planning, Team NYCasuals's program is the fruit of all members' technological and coding knowledge.

The programming approach consists of a few steps:
1) Brainstorm a concept
2) Write pseudocode
3) Write the actual code
4) Compile the code and execute the program
5) Test the program in CsBot Rescue
6) Debug the code 7) Optimize the code

IV. STRATEGIES

In general, NYCasuals's Artificial Intelligence (AI) uses a series of basic and complex strategies that involve navigating the robot around the map to gain points as efficiently as possible. Basic strategies are implemented using the CSRSim AI Development Panel. They include detecting all objects, avoiding map borders, avoiding traps, avoiding walls and speeding up in swamps. Complex strategies require coding in C using Code::Blocks and Notepad++. Examples are line tracing, sweeping movement, RRGGBB collection, staying in special zone, angle turning, wall tracing and path-finding using the A* search algorithm.

A. Detecting Objects

The robot's colour sensors are used to seek nearby objects. When an object is found, the robot stops moving and flashes its LED for 3 seconds to collect the object.

B. Avoiding Yellow Border

When the colour sensors detect the yellow borders in World 1, the wheel values are set to negative so that the robot moves backwards.

C. Avoiding Traps

In World 1, the colour sensors are used to detect boundaries. In World 2, the X and Y coordinates indicate the location of the traps. Wheel values are set to negative to avoid traps.

D. Avoiding Walls

When the ultrasonic sensors detect a wall,the wheel values are set to negative so that the robot moves backwards.

E. RRGGBB collection

When a robot collects 2 red, 2 green and 2 black objects, it gains 180 bonus points. Our program aims to capitalise on this by collecting at most 2 of each.

F. Sweeping

The chances of the robot encountering an object is increased using a âĂŸsweeping' function, where the robot's wheels have different speeds at each interval. This allows the robot to move in a âĂŸleft-right' manner which is much more efficient than simply moving straight forward. This is implemented using the robot's compass and a separate variable.

G. Staying in Special Zone

In special zones, points gained are doubled. Our program enables the robot to stay inside the special zone using colour sensors and maximise points acquired.

H. Wall Tracing

By using the robot's ultrasonic sensors, we can program it to follow walls. This is only applicable to certain maps, such as one with many objects around a wall.

I. Avoiding Borders in World 2

We make use of the X and Y coordinates, as well as the robot's compass to avoid the World 2 borders.

J. Angle Turning

With a destination angle in mind, the robot turns either clockwise or anticlockwise (whichever is shorter) to it. This is coded using if - else statements.

```
if Compass>=targetAngle then
    CWAngleDiff := Compass-targetAngle
    ACWAngleDiff := 360-(Compass-targetAngle)
else
    ACWAngleDiff := targetAngle-Compass
    CWAngleDiff := 360-(targetAngle-Compass)

if CWAngleDiff<=ACWAngleDiff then
    speed(maxSpeed,minSpeed)
else if ACWAngleDiff<CWAngleDiff then
    speed(minSpeed,maxSpeed)
```

```
else
    speed(maxSpeed,maxSpeed)

where minSpeed and maxSpeed are
predefined integers from -5 to 5
```

K. Bitmapping

The Python Imaging Library (PIL) is used to convert a BMP file into an array of numbers with map details. (See Fig.1 and Fig.2)

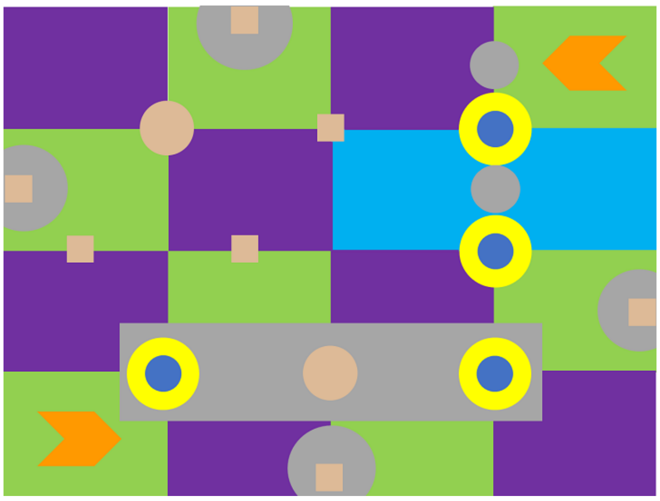

Fig. 1. The original map (720x540)

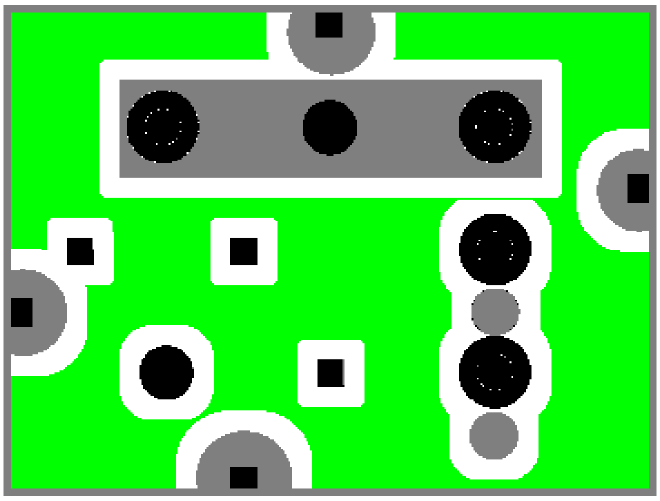

Fig. 2. The map (360x270) processed using PIL. It is flipped vertically as the point (0,0) starts at the top left hand corner for image files. Movement costs from adjacent and diagonal squares are increasing from green to white to grey to black. In the array, green=0, white=1, grey=2 and black=3.

L. A* Search Algorithm

The A* algorithm is used on the generated grid to determine the Single-Source Shortest Path (SSSP) from a source node to a destination node.

1) Heuristics: The octile heuristic is used as the map is represented in a 2d grid.

```
function heuristic(node)
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D)
        * min(dx, dy)
```

2) Achieving smooth straight-line movement: A standard A* search generates undesirable "zigzag" effects (Fig.3). To smoothen the path, direction-change penalties are implemented into the navigation cost.

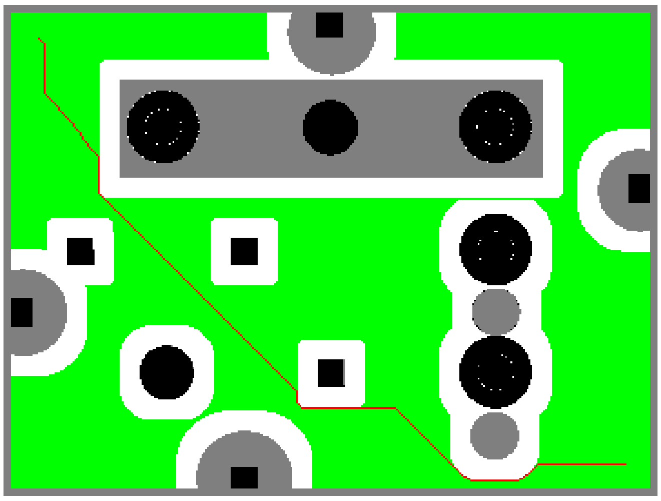

Fig. 3. Bad path with the "zigzag" effect generated by the A* algorithm with the octile heuristic.

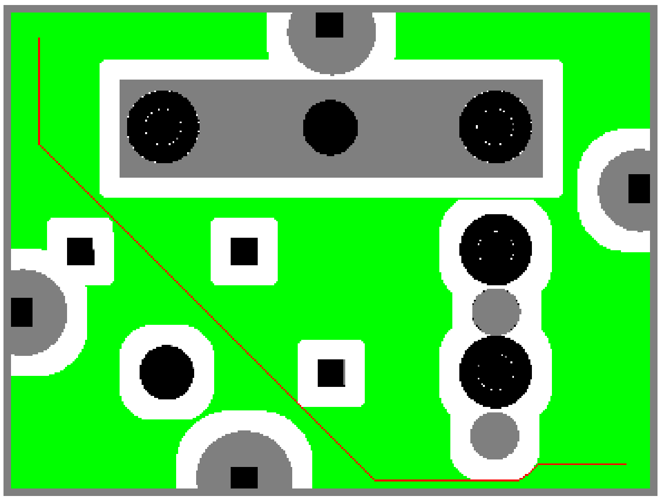

Fig. 4. Good path without the "zigzag" effect generated by the A* algorithm with direction-change penalties.

M. Trigonometry

Trigonometry is used to calculate turning angles for the robot.

In NYCasuals's program, theta (θ) is defined as the target angle which the robot needs to turn to in order to reach a certain destination. Trigonometry is used to initialise the value of theta so as to direct the robot towards different target destinations in World 2 using the compass. In this case, the function **atan** (arctangent) in C/C++ is used to determine theta, as the length of the perpendicular sides of the triangle are already known. The value of theta is used to direct the robot towards the different destination nodes. It is also used to move directly to the Super Object.

```
function trigo (targetX, targetY)
    diffX := targetX - robotX
    diffY := targetY - robotY
    angle := atan(diffY/diffX)
```

V. CONCLUSION

This document describes the tactics and strategies that Team NYCasuals uses in the RoboCupJunior Rescue CoSpace competition. As the world champion in 2016 and 2017, Team NYCasuals will do its best defend its title in RoboCup 2018 Montreal, Canada.

REFERENCES

[1] "Rescue," rcj.robocup.org. [Online]. Available: http://rcj.robocup.org/rescue.html. [Accessed: 2018].

[2] R. Belwariar, âĂIJA* Search Algorithm,âĂİ GeeksforGeeks, 21-May-2018. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/. [Accessed: 12-Jun-2018].

[3] A. Patel, âĂIJAmit's A* Pages,âĂİ AmitâĂŹs A* Pages. [Online]. Available: https://theory.stanford.edu/ amitp/GameProgramming/. [Accessed: 12-Jun-2018].

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction To Algorithms, 3rd ed. Cambridge, Massachusetts: The MIT Press, 2009.

[5] A. Nash, âĂIJTheta*: Any-Angle Path Planning for Smoother Trajectories in Continuous Environments,âĂİ AiGameDev.com, 08-Sep-2010. [Online]. Available: https://aigamedev.com/open/tutorials/theta-star-any-angle-paths/. [Accessed: 17-Jun-2018].