AN AIR DEFENCE SCENARIO SIMULATION WITH SWARM ROBOTS

Burak Eren Dere
Department of Computer Engineering,
Middle East Technical University,
06531 Ankara, Turkey
Email: eren.dere@metu.edu.tr

Abstract

An air defence scenario is being implemented by using an open source 2d physics engine. There are swarm robots patrolling an area and trying to catch incoming rockets by estimating their trajectories. Swarm robots will try to mimic the behavior of flocks. Each robot will follow a modified boids algorithm. When I submitted progress report, scenario was not completed. A pure boids algorithm was implemented and robots were moving freely. In this part of the project, boids algorithm has been modified with rocket interception and refueling circumstances. And scenario is completed.

Keywords

Swarm intelligence, Boids Algorithm, aggregate motion, path planning

I. Introduction

During this project, I was working on an air defence simulation with swarm robots. According to the scenario, there are two teams(blue and red) which try to destroy eachother's bases by sending rockets. Team 1(blue) has a robot swarm which patrols nearby its base and protects it from incoming rockets by intercepting them which has high potential threat. Robots try to mimic the behavior of birds.

II. Background and Related Work

In this project, I have used a 2D physics engine called Box2D[1], I tried to make physical movements as realistic as possible by making physical calculations on every move. For that, BoX2D became very useful. And for representing the simulation, I have used a multimedia library called SFML[2] which uses OpenGL. I programmed this project with C++ language which I made use of its object oriented property. During my implementation I found Craig Reynold's paper(1987)[3] which introduces boids algorithm for my swarm robot implementation. That study explains aggregate motion of flock birds or school of fish. Boids algorithm simulates flocking behavior of birds. In this algorithm, velocities of swarm members depend on various rules. Simples rules are separation for avoiding local flockmates, alignment for steering towards the average heading of local flockmates and cohesion for moving towards the average position of local flockmates.

III. Project Work

Since I worked on continuous domain, in order to solve collisions in each step, I had to determine my entities and their shapes and implement them both in SFML and Box2D. I implemented Robot, Rocket and Base as my entities in this study. And then, I implemented behavior of robots. Robots have bunch of choices during simulation, they can move as a swarm, chase a robot which has dangerous estimated trajectory, or refuel if they have little fuel left.

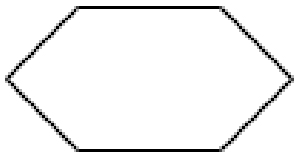

Fig. 1: Shape of Robot.

Robots are hexagonal shapes and their center of mass is at the middle of the shape. During the simulation, gravity is being applied to robots as a constant external force. At the same time, if robots collide with rockets from enemy team or particles coming from exploded entities, they also result as an external force on robots. In my implementation, Robots only collide with enemy Rockets and Particles. External forces are likely to disturb the balance of robots, causing them spin because of the torque created by external forces. In each time step of the simulation, robots execute $orientationControl()$ and $gravityNeutralizer$ steps in order to ensure balance during operations.

Robots have two sensors called $rocketSensor$ and $rocketBSensor$. $rocketSensor$ is a circle shaped sensor with its center is aligned with robot's center of mass which senses friendly robots and saves their their position and velocities. This information is later used for swarm behavior with boids algorithm. $rocketBSensor$ is a bigger version of $rocketSensor$ and it senses up to $700m$ distance for incoming rockets. When this sensor sees an incoming rocket, it saves its velocity. This velocity is used by the robot to estimate its trajectory.

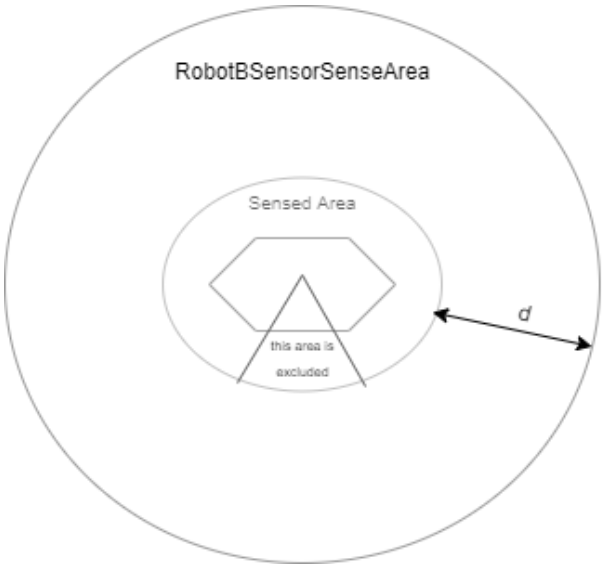

Fig. 2: $rocketSensor$ and $rocketBSensor$

I added 5 states to robot which are $Patrolling$,$Steady$,$Refueling$,$Chasing$ and $Returning$. In $Patrolling$ state, robot patrols nearby its base and tries to group up with robots it sees and forms a swarm. In $Steady$ state, robots stand still, in $Refueling$ state, robots go to base and refuel, in $Chasing$ state, robot chases rocket it decided to intercept after various calculations, and in $Returning$ state, robots return back to base

after successful interception.

For patrolling state, pseudo code of $patrolArea()$ function is showed below.

function PATROLAREA
  $dir \Leftarrow rand()\%2$ ▷ randomly select direction
  $pos \Leftarrow body.getPosition()$
  $vel \Leftarrow body.getLinearVelocity()$
  if isInArea() **then** ▷ check whether robot is in patrol area or not
    $turnDirection(dir)$ ▷ change direction randomly
  end if
  $desiredVelocity$
  if robotsInArea.size()¿0 **then** ▷ If there are robots nearby, compute swarm velocity
    $desiredVelocity = targetSpeed * direction$
    $desiredVelocity+ = boidsAlgorithmVelocity()$
    $desiredVelocity.Normalize()$ ▷ Normalization is required to make boids velocity better
    $desiredVelocity = targetSpeed * desiredVelocity$
  else
    $desiredVelocity = targetSpeed * direction$ ▷ If there are no robots, move freely
  end if
  $velocityChange = desiredVelocity - vel$
  $force = body.getMass() * 240.f * velocityChange$ ▷ 240.f is a constant I came up with
  $velocityChange.Normalize()$
  $body.ApplyForce(force, body.getWorldCenger())$ ▷ Apply force to robot's center of mass
  $fuel- = 0.02$ ▷ Reduce fuel for this operation
end function

The implementation of $boidsAlgorithmVelocity()$ was done in my progress report[4]. Robot act() function which is called every time step is shown in pseudecode below.

function ACT
  $orientationControl()$ ▷ this is for balance of robots
  $gravityNeutralizer$ ▷ neutralize gravity acting on robot
  if state == State::$Patrolling$ **then**
    $patrolArea()$
  else if state == State::$Chasing$ **then**
    $intercept()$ ▷ go intercept rocket with high threat
  else if state == State::$Refueling$ **then**
    $refuel()$ ▷ simply go base and refuel
  else if state == State::$Steady$ **then**
    $noop$
  else if state == State::$Returning$ **then**
    $returnBase()$ ▷ returns to base
  end if
end function

When $robotBSensor$ senses rockets and checks their trajectories, I use equation $p(n) = p_0 + nv + \frac{(n^2+n)a}{2}$ for trajectory estimation. In this equation a is gravity acceleration and n is the time step.

After implementing robots, I implemented rocket shape,

Rockets are fired from guns mounted on top of bases. Base shape is shown below,

Fig. 3: Shape of Rocket.

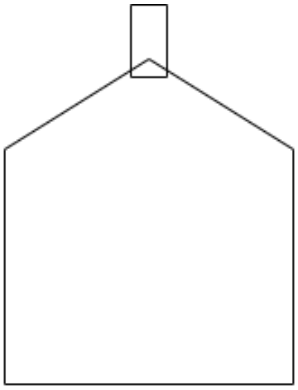

Fig. 4: Shape of Base.

A. *Problem Description*

In this study, behavior of swarm robots which try to intercept incoming rockets is being examined. An instance of our problem is composed of two teams T_1 and T_2. Both teams have bases B_1 and B_2 respectively which fire up rockets roc_x. T_1 has N robots $R = R_1, ..., R_N$ which guard B_1 from incoming rockets fired from B_2. Each robot R_i has two sensors named S_1 and S_2. S_1 senses friendly robots for swarm behavior and S_2 senses for incoming rockets.A robot R_i is assigned to perform:

- Patrolling with nearby rockets in an area
- Sensing for incoming rockets and try to intercept dangerous ones if less than 4 robots are already chasing it.
- Managing their fuel situation and refuel if they are running out of fuel.

B. *Method*

Different combinations of parameters have been explored for air defence simulation. These are namely:

- Sensor range s_r
- Robot count r_c
- Rocket firerate f_r, determined by $\frac{rocketCount}{seconds}$

With these parameters, air defence scenario outcomes are examined. There are two outcomes namely, T_1 wins or T_2 wins.

C. Evaluation

An air defence scenario is examined as a case study. In this scenario there are 2 teams(blue and red). Each team fires up rockets towards enemy bases. Blue team has swarm robots protecting its base, on the other hand red team does not have swarm guards.With changing s_r, r_c and f_r, outcomes of simulations are evaluated. In order to calculate effectiveness of robots of blue team, win rate w_r of T_1 is calculated for each parameter and derive relations. I tweaked these parameters separately and examined win rates. Other parameters remained constant at that time.

I have found a correlation between sensor range s_r and win rate. If robots can see incoming rockets earlier, they act earlier to intercept them which increases the win rate. However, giving robots more than 600 meters sensor radius does not change the win rate because it reaches to the peak.

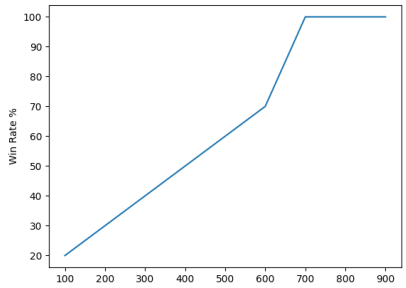

Fig. 5: Range of $sensorB$ - Win rate.

There is also a correlation between rocket count r_c and win rate. More than 32 robots is enough for 100% winrate.

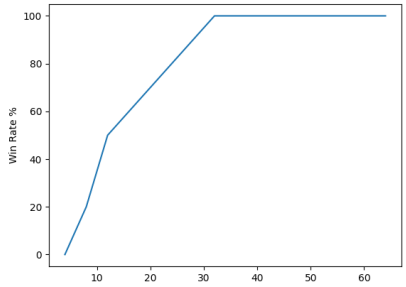

Fig. 6: Robot count - Win rate.

After examining firerate of rockets from bases, I found no correlation if other parameters are kept constant. Win rate keeps fluctuating.

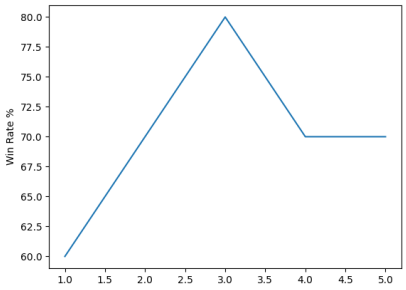

Fig. 7: Firerate - Win rate .

IV. Implementation Details

I implemented this project in C++ with my 3.00GHz Intel Core i7-9700F CPU and 16GB ram. I experienced fps drops when there were lots of particles, robots and rockets in the scene to render. I have used object oriented properties of c++. Windows-MinGW as my toolchain. I have also tested my code in my ubuntu virtual machine and it worked. After posting my progress report, I decided to make this project as independent from platforms as possible, so I turned it into a cmake project. If there are compiled libraries of SFML and Box2D, it should compile without errors with cmake.There were some implementations of boids algorithm with various programming languages, however I could not find any project implementing a specific scenario like air defence. Now there is one implementation with C++ language. I posted my project repository on Github[5] in order to share my effort with friends and collegues who are interested in implementing 2D physics simulators with Box2D. There are fine examples of collision listening, shape designing, fixture and joint designing, particle explosions, dead body cleaning, trajectory estimation and an extended boids algorithm with air defence specific goals which I have implemented.

V. Conclusions

At the beginning of my work, I learned the basics of SFML and Box2D physics engine. With entity and factory classes I implemented, I merged SFML and Box2D shapes so this project can also act as a SFML,Box2D wrap library for simplicity. I started implementing the shapes and base classes for my implementation, in the middle of the term, I managed to implement pure boids algorithm to my robots but my project was lacking complete air defence scenario. After that, I started extending the boids algorithm with air defence specific goals and completed. In my github repository, I also added an interception example taken from my simulation. In the future, I can turn this project into a game and add new ai features like swarm attack etc.

References

[1] "Box2d," https://box2d.org/, accessed: 2020-06-15.

[2] "Sfml," https://en.wikipedia.org/wiki/Simple$_{a}nd_{F}ast_{M}ultimedia_{L}ibrary, accessed$: 2020 − 06 − 15.

[3] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[4] "Boidsimplementation," https://github.com/yatiyr/SwarmBattle/blob/master/BurakErenDere$_{P}rogressReport.pdf, accessed$: 2020 − 06 − 15.

[5] "Githubrepo," https://github.com/yatiyr/SwarmBattle, accessed: 2020-06-15.