Introduction to AI Assignment 2: Search Agent Simple 2D Ball Game

By

資工二乙 408262416

陳嬿婷

Date Submitted: April 5th,

2021

Writing

1. **Describe the goal formulation and problem formulation your team used.**

   - Goal formulation :

     my agent變成前三名

   - Problem formulation

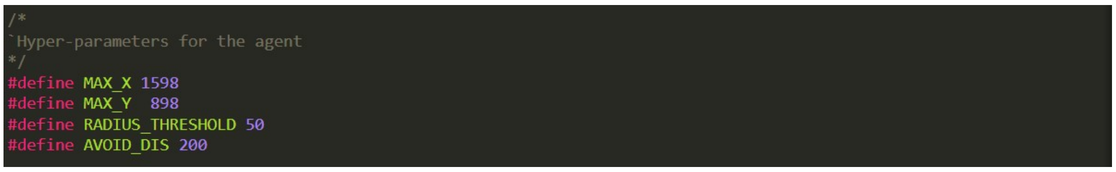
     排優先權，已得到最高勝率

2. **Describe the search strategy your team used, and explain the design concept.**

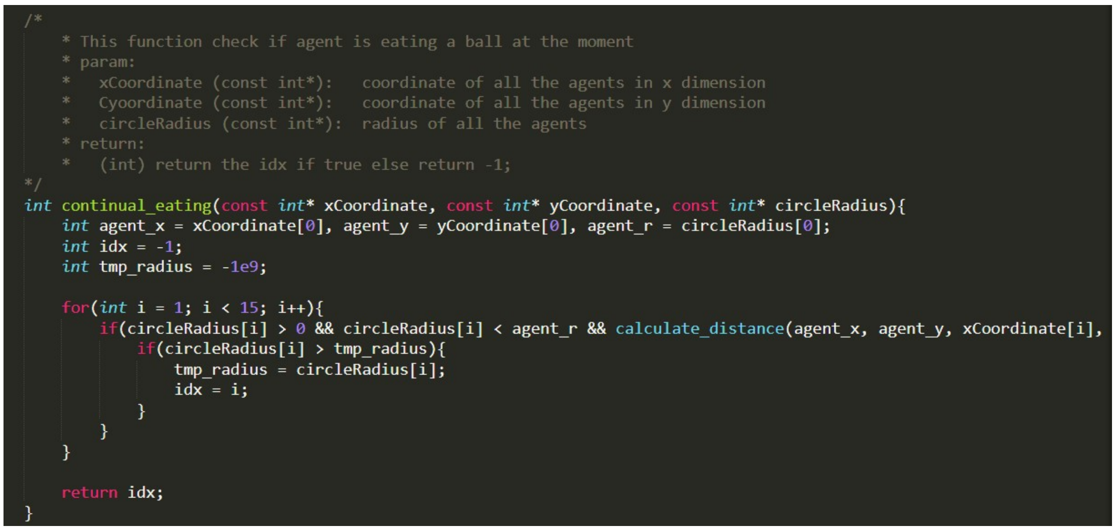
   - Search strategy :

     排優先權 1. 避免撞牆 2. 遠離大球 3. 決定target。所以狀況皆已前進方向以及距離來判斷，若加速方向與前進方向相同時會給一個反向力，讓球速減緩，反之則不變，使得球能準確地抵達目標位置，以達到最高效率。當球小於50時，會先以黑球為主，大於等於50會開始找離自己最近最大且比自己小的球做為目標，且會將他吃完後才會找尋下一個目標，不過會因為要避免撞牆、大球等，而改變目標。

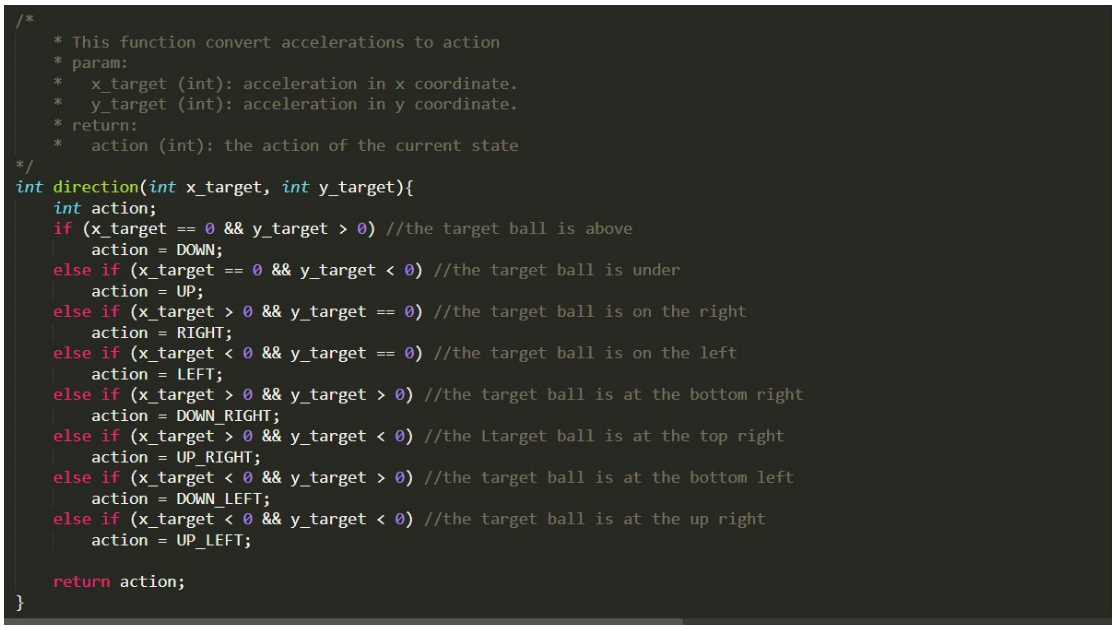
   - Design concept :



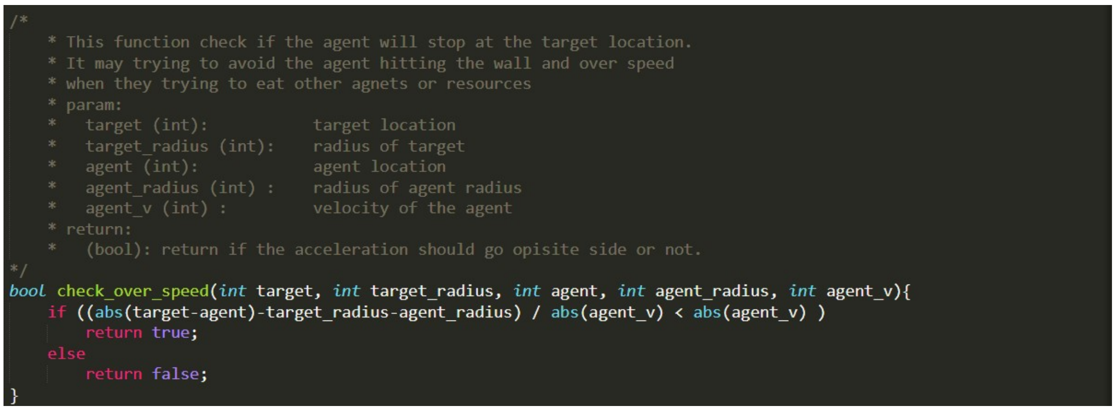
```
/*
`Hyper-parameters for the agent
*/
#define MAX_X 1598
#define MAX_Y  898
#define RADIUS_THRESHOLD 50
#define AVOID_DIS 200
```



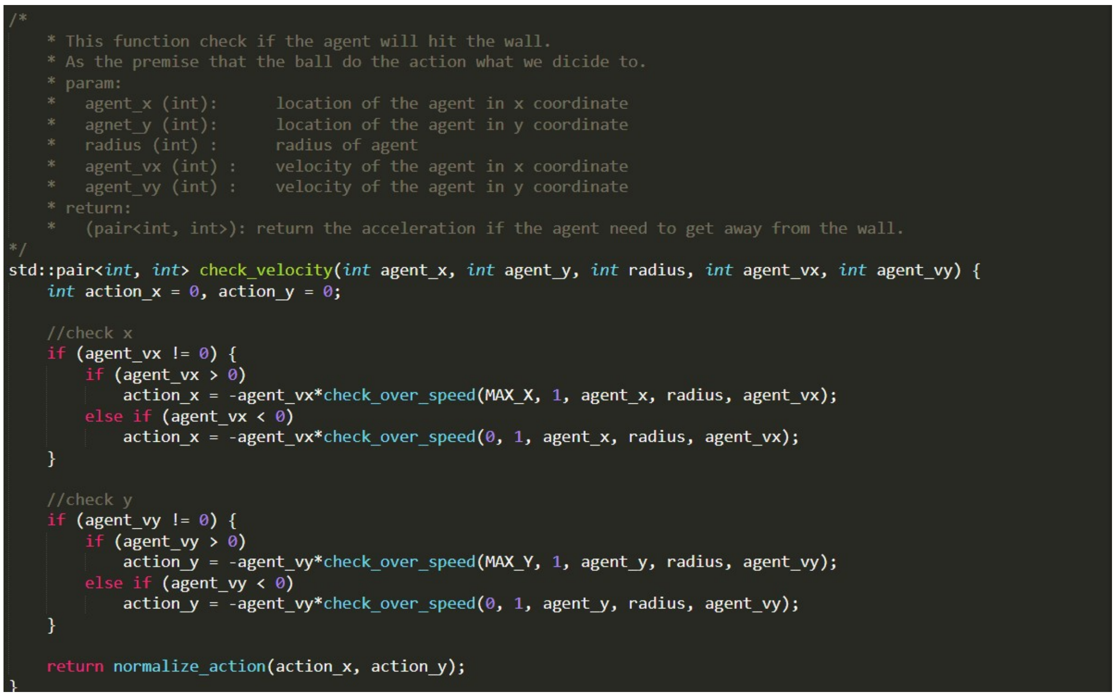
```
/*
    * This function check if agent is eating a ball at the moment
    * param:
    *   xCoordinate (const int*):   coordinate of all the agents in x dimension
    *   Cyoordinate (const int*):   coordinate of all the agents in y dimension
    *   circleRadius (const int*):  radius of all the agents
    * return:
    *   (int) return the idx if true else return -1;
*/
int continual_eating(const int* xCoordinate, const int* yCoordinate, const int* circleRadius){
    int agent_x = xCoordinate[0], agent_y = yCoordinate[0], agent_r = circleRadius[0];
    int idx = -1;
    int tmp_radius = -1e9;

    for(int i = 1; i < 15; i++){
        if(circleRadius[i] > 0 && circleRadius[i] < agent_r && calculate_distance(agent_x, agent_y, xCoordinate[i],
            if(circleRadius[i] > tmp_radius){
                tmp_radius = circleRadius[i];
                idx = i;
            }
        }
    }

    return idx;
}
```



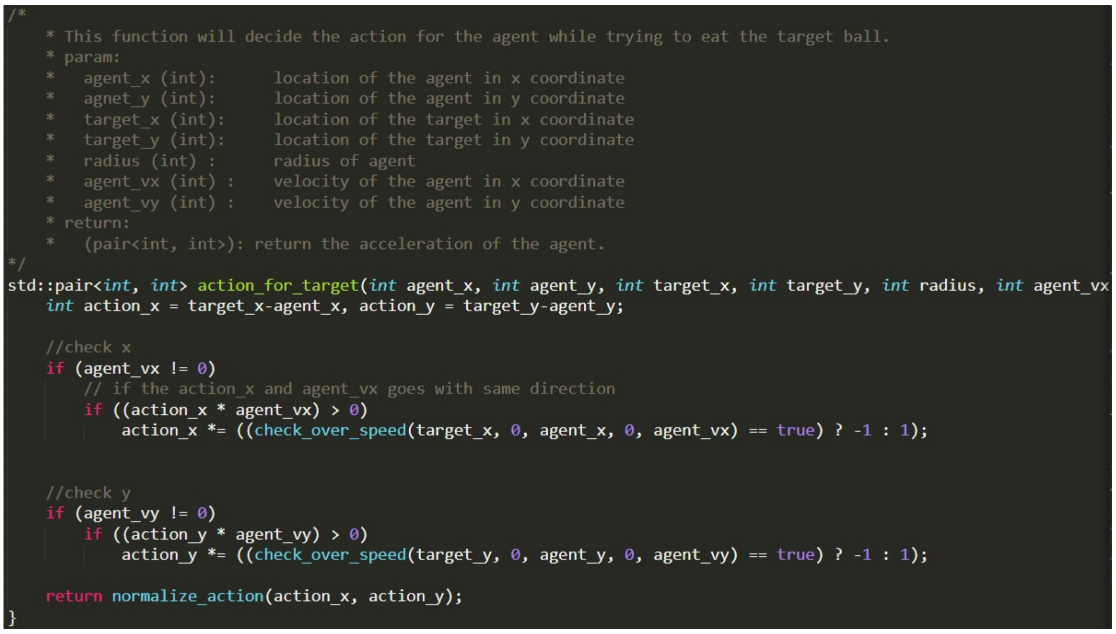
```
/*
    * This function convert accelerations to action
    * param:
    *   x_target (int): acceleration in x coordinate.
    *   y_target (int): acceleration in y coordinate.
    * return:
    *   action (int): the action of the current state
*/
int direction(int x_target, int y_target){
    int action;
    if (x_target == 0 && y_target > 0) //the target ball is above
        action = DOWN;
    else if (x_target == 0 && y_target < 0) //the target ball is under
        action = UP;
    else if (x_target > 0 && y_target == 0) //the target ball is on the right
        action = RIGHT;
    else if (x_target < 0 && y_target == 0) //the target ball is on the left
        action = LEFT;
    else if (x_target > 0 && y_target > 0) //the target ball is at the bottom right
        action = DOWN_RIGHT;
    else if (x_target > 0 && y_target < 0) //the Ltarget ball is at the top right
        action = UP_RIGHT;
    else if (x_target < 0 && y_target > 0) //the target ball is at the bottom left
        action = DOWN_LEFT;
    else if (x_target < 0 && y_target < 0) //the target ball is at the up right
        action = UP_LEFT;

    return action;
}
```



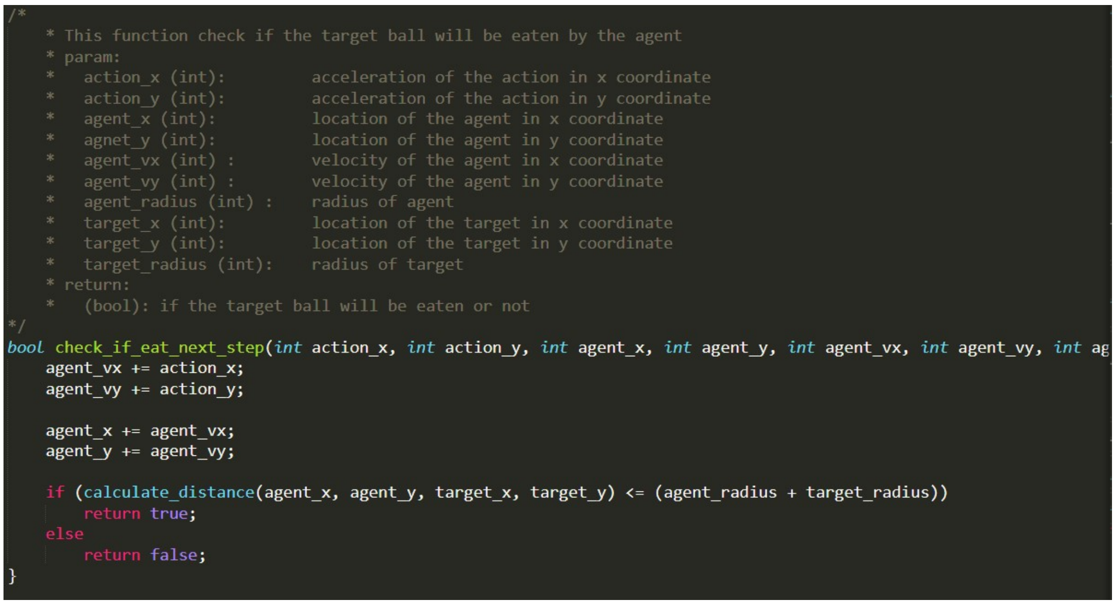
```
/*
    * This function check if the agent will stop at the target location.
    * It may trying to avoid the agent hitting the wall and over speed
    * when they trying to eat other agnets or resources
    * param:
    *   target (int):           target location
    *   target_radius (int):    radius of target
    *   agent (int):            agent location
    *   agent_radius (int) :    radius of agent radius
    *   agent_v (int) :         velocity of the agent
    * return:
    *   (bool): return if the acceleration should go opisite side or not.
*/
bool check_over_speed(int target, int target_radius, int agent, int agent_radius, int agent_v){
    if ((abs(target-agent)-target_radius-agent_radius) / abs(agent_v) < abs(agent_v) )
        return true;
    else
        return false;
}
```



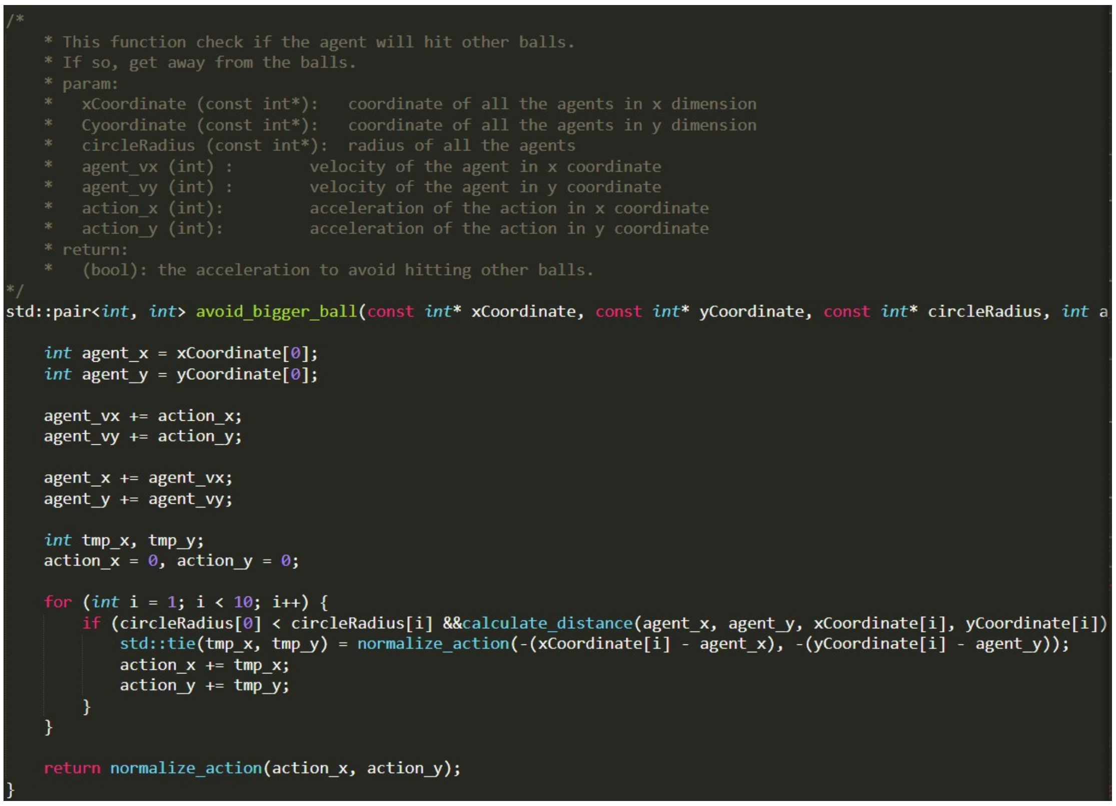
```cpp
/*
    * This function check if the agent will hit the wall.
    * As the premise that the ball do the action what we dicide to.
    * param:
    *   agent_x (int):      location of the agent in x coordinate
    *   agnet_y (int):      location of the agent in y coordinate
    *   radius (int) :      radius of agent
    *   agent_vx (int) :    velocity of the agent in x coordinate
    *   agent_vy (int) :    velocity of the agent in y coordinate
    * return:
    *   (pair<int, int>): return the acceleration if the agent need to get away from the wall.
*/
std::pair<int, int> check_velocity(int agent_x, int agent_y, int radius, int agent_vx, int agent_vy) {
    int action_x = 0, action_y = 0;

    //check x
    if (agent_vx != 0) {
        if (agent_vx > 0)
            action_x = -agent_vx*check_over_speed(MAX_X, 1, agent_x, radius, agent_vx);
        else if (agent_vx < 0)
            action_x = -agent_vx*check_over_speed(0, 1, agent_x, radius, agent_vx);
    }

    //check y
    if (agent_vy != 0) {
        if (agent_vy > 0)
            action_y = -agent_vy*check_over_speed(MAX_Y, 1, agent_y, radius, agent_vy);
        else if (agent_vy < 0)
            action_y = -agent_vy*check_over_speed(0, 1, agent_y, radius, agent_vy);
    }

    return normalize_action(action_x, action_y);
}
```



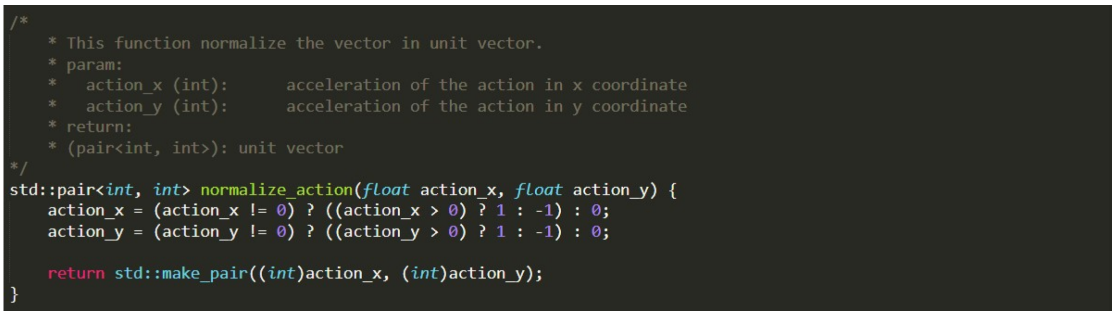
```cpp
/*
    * This function will decide the action for the agent while trying to eat the target ball.
    * param:
    *   agent_x (int):      location of the agent in x coordinate
    *   agnet_y (int):      location of the agent in y coordinate
    *   target_x (int):     location of the target in x coordinate
    *   target_y (int):     location of the target in y coordinate
    *   radius (int) :      radius of agent
    *   agent_vx (int) :    velocity of the agent in x coordinate
    *   agent_vy (int) :    velocity of the agent in y coordinate
    * return:
    *   (pair<int, int>): return the acceleration of the agent.
*/
std::pair<int, int> action_for_target(int agent_x, int agent_y, int target_x, int target_y, int radius, int agent_vx
    int action_x = target_x-agent_x, action_y = target_y-agent_y;

    //check x
    if (agent_vx != 0)
        // if the action_x and agent_vx goes with same direction
        if ((action_x * agent_vx) > 0)
            action_x *= ((check_over_speed(target_x, 0, agent_x, 0, agent_vx) == true) ? -1 : 1);


    //check y
    if (agent_vy != 0)
        if ((action_y * agent_vy) > 0)
            action_y *= ((check_over_speed(target_y, 0, agent_y, 0, agent_vy) == true) ? -1 : 1);

    return normalize_action(action_x, action_y);
}
```



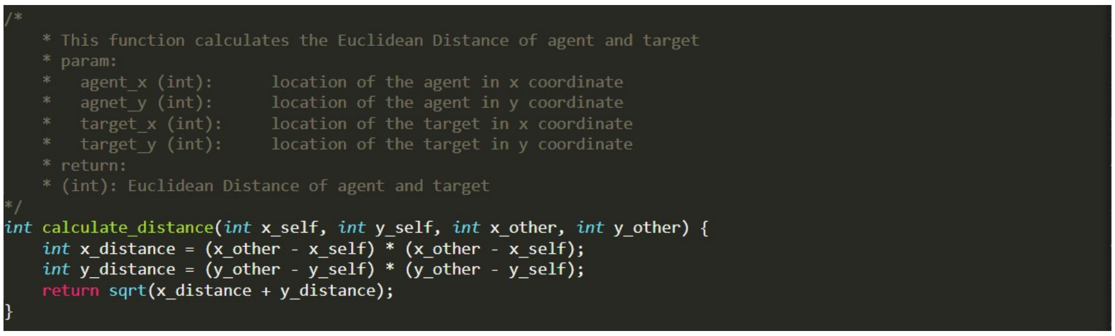
```cpp
/*
    * This function check if the target ball will be eaten by the agent
    * param:
    *   action_x (int):         acceleration of the action in x coordinate
    *   action_y (int):         acceleration of the action in y coordinate
    *   agent_x (int):          location of the agent in x coordinate
    *   agnet_y (int):          location of the agent in y coordinate
    *   agent_vx (int) :        velocity of the agent in x coordinate
    *   agent_vy (int) :        velocity of the agent in y coordinate
    *   agent_radius (int) :    radius of agent
    *   target_x (int):         location of the target in x coordinate
    *   target_y (int):         location of the target in y coordinate
    *   target_radius (int):    radius of target
    * return:
    *   (bool): if the target ball will be eaten or not
*/
bool check_if_eat_next_step(int action_x, int action_y, int agent_x, int agent_y, int agent_vx, int agent_vy, int ag
    agent_vx += action_x;
    agent_vy += action_y;

    agent_x += agent_vx;
    agent_y += agent_vy;

    if (calculate_distance(agent_x, agent_y, target_x, target_y) <= (agent_radius + target_radius))
        return true;
    else
        return false;
}
```



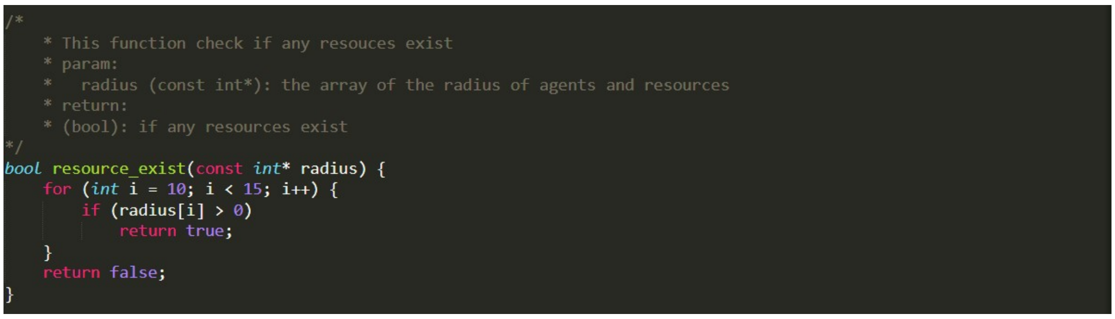
```cpp
/*
    * This function check if the agent will hit other balls.
    * If so, get away from the balls.
    * param:
    *   xCoordinate (const int*):   coordinate of all the agents in x dimension
    *   Cyoordinate (const int*):   coordinate of all the agents in y dimension
    *   circleRadius (const int*):  radius of all the agents
    *   agent_vx (int) :        velocity of the agent in x coordinate
    *   agent_vy (int) :        velocity of the agent in y coordinate
    *   action_x (int):         acceleration of the action in x coordinate
    *   action_y (int):         acceleration of the action in y coordinate
    * return:
    *   (bool): the acceleration to avoid hitting other balls.
*/
std::pair<int, int> avoid_bigger_ball(const int* xCoordinate, const int* yCoordinate, const int* circleRadius, int a

    int agent_x = xCoordinate[0];
    int agent_y = yCoordinate[0];

    agent_vx += action_x;
    agent_vy += action_y;

    agent_x += agent_vx;
    agent_y += agent_vy;

    int tmp_x, tmp_y;
    action_x = 0, action_y = 0;

    for (int i = 1; i < 10; i++) {
        if (circleRadius[0] < circleRadius[i] &&calculate_distance(agent_x, agent_y, xCoordinate[i], yCoordinate[i])
            std::tie(tmp_x, tmp_y) = normalize_action(-(xCoordinate[i] - agent_x), -(yCoordinate[i] - agent_y));
            action_x += tmp_x;
            action_y += tmp_y;
        }
    }

    return normalize_action(action_x, action_y);
}
```



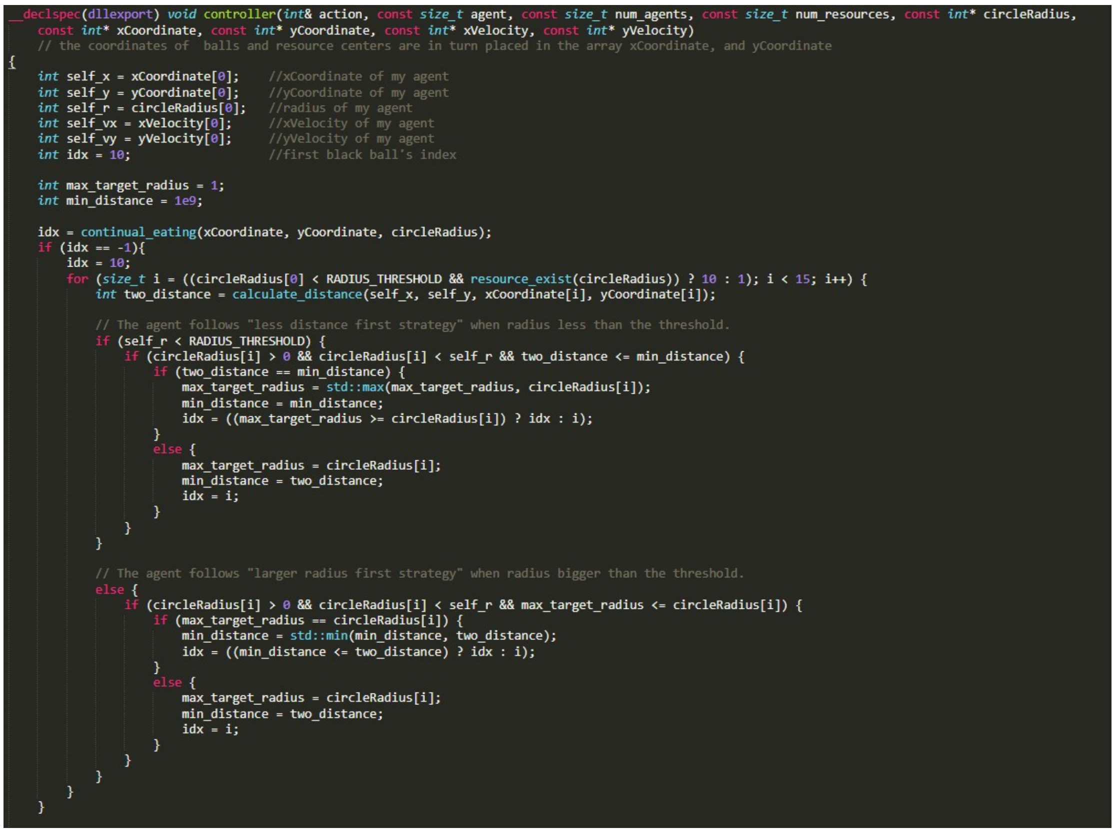
```cpp
/*
    * This function normalize the vector in unit vector.
    * param:
    *   action_x (int):     acceleration of the action in x coordinate
    *   action_y (int):     acceleration of the action in y coordinate
    * return:
    * (pair<int, int>): unit vector
*/
std::pair<int, int> normalize_action(float action_x, float action_y) {
    action_x = (action_x != 0) ? ((action_x > 0) ? 1 : -1) : 0;
    action_y = (action_y != 0) ? ((action_y > 0) ? 1 : -1) : 0;

    return std::make_pair((int)action_x, (int)action_y);
}
```



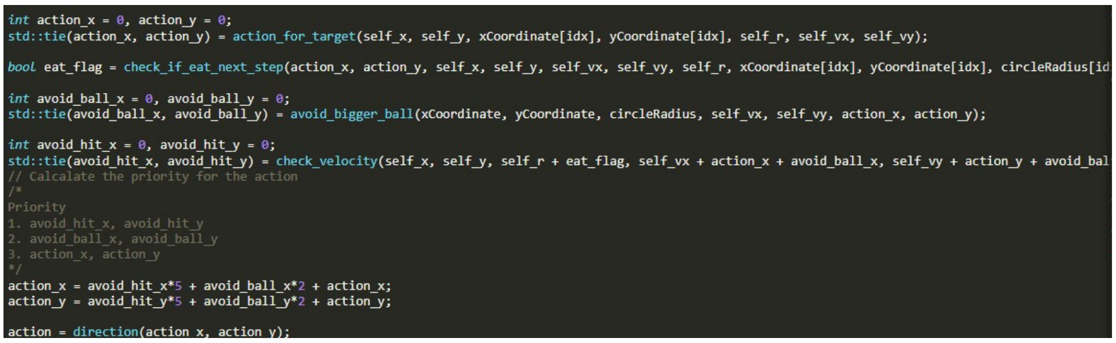
```cpp
/*
    * This function calculates the Euclidean Distance of agent and target
    * param:
    *   agent_x (int):      location of the agent in x coordinate
    *   agnet_y (int):      location of the agent in y coordinate
    *   target_x (int):     location of the target in x coordinate
    *   target_y (int):     location of the target in y coordinate
    * return:
    * (int): Euclidean Distance of agent and target
*/
int calculate_distance(int x_self, int y_self, int x_other, int y_other) {
    int x_distance = (x_other - x_self) * (x_other - x_self);
    int y_distance = (y_other - y_self) * (y_other - y_self);
    return sqrt(x_distance + y_distance);
}
```



```cpp
/*
    * This function check if any resouces exist
    * param:
    *   radius (const int*): the array of the radius of agents and resources
    * return:
    * (bool): if any resources exist
*/
bool resource_exist(const int* radius) {
    for (int i = 10; i < 15; i++) {
        if (radius[i] > 0)
            return true;
    }
    return false;
}
```



```cpp
__declspec(dllexport) void controller(int& action, const size_t agent, const size_t num_agents, const size_t num_resources, const int* circleRadius,
    const int* xCoordinate, const int* yCoordinate, const int* xVelocity, const int* yVelocity)
    // the coordinates of  balls and resource centers are in turn placed in the array xCoordinate, and yCoordinate
{
    int self_x = xCoordinate[0];    //xCoordinate of my agent
    int self_y = yCoordinate[0];    //yCoordinate of my agent
    int self_r = circleRadius[0];   //radius of my agent
    int self_vx = xVelocity[0];     //xVelocity of my agent
    int self_vy = yVelocity[0];     //yVelocity of my agent
    int idx = 10;                   //first black ball's index

    int max_target_radius = 1;
    int min_distance = 1e9;

    idx = continual_eating(xCoordinate, yCoordinate, circleRadius);
    if (idx == -1){
        idx = 10;
        for (size_t i = ((circleRadius[0] < RADIUS_THRESHOLD && resource_exist(circleRadius)) ? 10 : 1); i < 15; i++) {
            int two_distance = calculate_distance(self_x, self_y, xCoordinate[i], yCoordinate[i]);

            // The agent follows "less distance first strategy" when radius less than the threshold.
            if (self_r < RADIUS_THRESHOLD) {
                if (circleRadius[i] > 0 && circleRadius[i] < self_r && two_distance <= min_distance) {
                    if (two_distance == min_distance) {
                        max_target_radius = std::max(max_target_radius, circleRadius[i]);
                        min_distance = min_distance;
                        idx = ((max_target_radius >= circleRadius[i]) ? idx : i);
                    }
                    else {
                        max_target_radius = circleRadius[i];
                        min_distance = two_distance;
                        idx = i;
                    }
                }
            }

            // The agent follows "larger radius first strategy" when radius bigger than the threshold.
            else {
                if (circleRadius[i] > 0 && circleRadius[i] < self_r && max_target_radius <= circleRadius[i]) {
                    if (max_target_radius == circleRadius[i]) {
                        min_distance = std::min(min_distance, two_distance);
                        idx = ((min_distance <= two_distance) ? idx : i);
                    }
                    else {
                        max_target_radius = circleRadius[i];
                        min_distance = two_distance;
                        idx = i;
                    }
                }
            }
        }
    }
```



```
int action_x = 0, action_y = 0;
std::tie(action_x, action_y) = action_for_target(self_x, self_y, xCoordinate[idx], yCoordinate[idx], self_r, self_vx, self_vy);

bool eat_flag = check_if_eat_next_step(action_x, action_y, self_x, self_y, self_vx, self_vy, self_r, xCoordinate[idx], yCoordinate[idx], circleRadius[id

int avoid_ball_x = 0, avoid_ball_y = 0;
std::tie(avoid_ball_x, avoid_ball_y) = avoid_bigger_ball(xCoordinate, yCoordinate, circleRadius, self_vx, self_vy, action_x, action_y);

int avoid_hit_x = 0, avoid_hit_y = 0;
std::tie(avoid_hit_x, avoid_hit_y) = check_velocity(self_x, self_y, self_r + eat_flag, self_vx + action_x + avoid_ball_x, self_vy + action_y + avoid_bal
// Calcalate the priority for the action
/*
Priority
1. avoid_hit_x, avoid_hit_y
2. avoid_ball_x, avoid_ball_y
3. action_x, action_y
*/
action_x = avoid_hit_x*5 + avoid_ball_x*2 + action_x;
action_y = avoid_hit_y*5 + avoid_ball_y*2 + action_y;

action = direction(action_x, action_y);
```

3. **Describe the challenges encountered when designing the agent.**

   如果起始在四個角落，且角落旁的球離黑點的距離較近，以至於會有比自己球大的可能性，所以會有落敗的情形。

4. **Give two scores from 1 to 10 to evaluate the performance of your teammates in this assignment in terms of Design and Implementation.**

   - 資工四乙 406262436 許承文 : 10
   - 資工二乙 408262349 張宇青 : 7