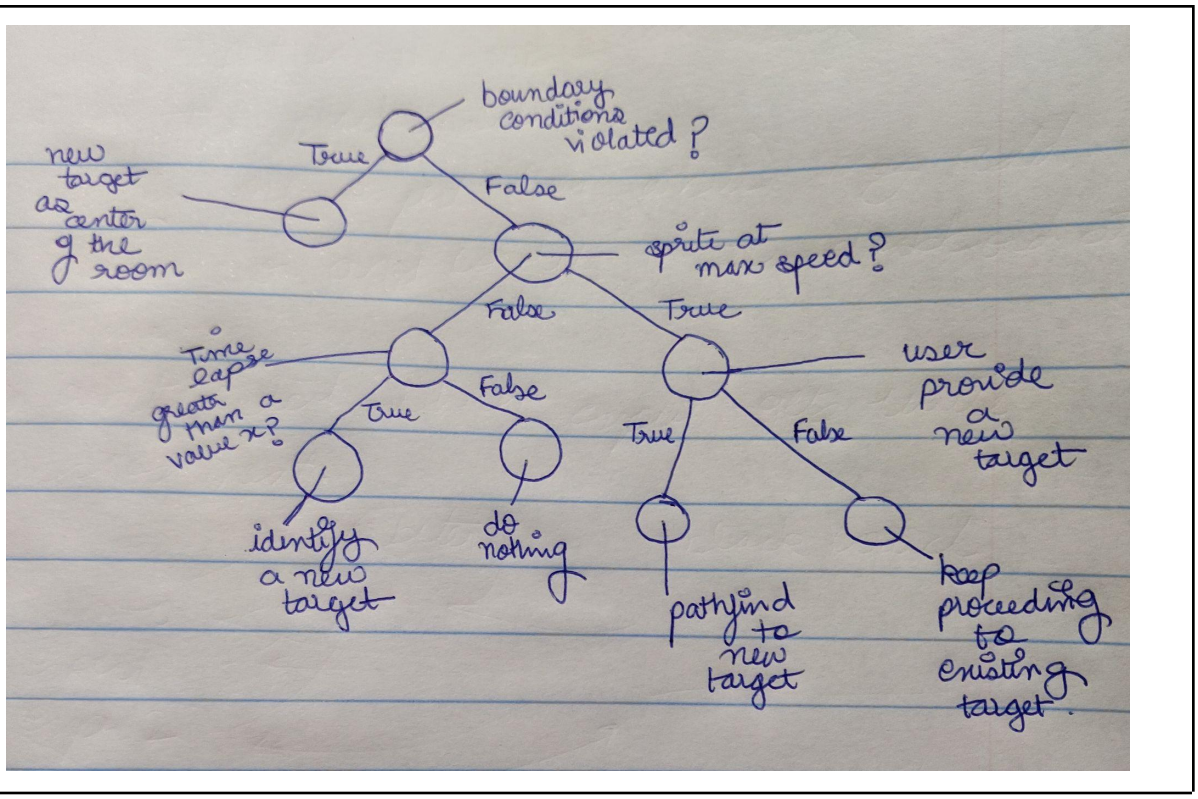
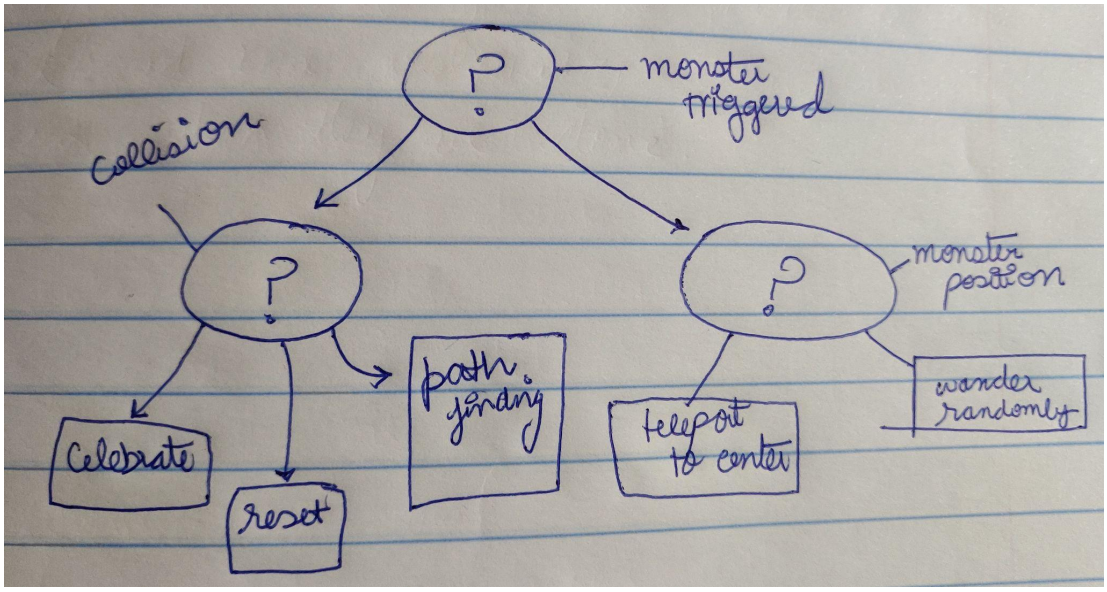
Assignment 4

The first task of this assignment was to implement a decision tree. To implement the tree, I decided to use the binary tree implementation in C++ using a structure and pointers to left and right nodes. The program will always start with the sprite moving towards a point outside the boundary conditions so as to trigger the first node (boundary condition violation check) of the decision tree. The value of each node would contain a condition. Based on this condition a function will be called and a true or false value would be returned. Using this true/false value, we either navigate to the left (for true) or right (for false). For most part, in order to ensure binary output we will have leaf nodes on the right and other conditions on the left. Least nodes always call a certain function associated with the action. For the purpose of this assignment, the entire screen acts as a room and we are not allowed to go beyond the boundary of this room. The decision tree starts at the root node, at which we check for this boundary condition. If evaluated to true, as suggested in the assignment, we always move towards the center of the room. Next condition checks for the speed of the sprite. In case it is at the maximum speed, it will switch to wander and move towards randomly generated points such that the x and y values can vary between -1000 and +1000 to ensure that the boundary conditions are violated. If the speed is decreasing, this would mean we are approaching the target point in which case, after a certain time interval we switch to a new randomly generated point. The last node checks for the user provided target to switch to pathfinding. If the mouse is clicked at any point on the screen, the sprite would stop the wandering behavior and start to move towards that point via a pathfinding algorithm. After it reaches the user provided point it switches back to the wander behavior to random points until a new target is provided by the user. This continues until the program is closed. At each point, we need to let the decision tree know of the target and character attributes. The decision tree is made aware of the current character position, the velocity, the target (randomly generated), the mouse pointer location and finally the current lapsed time. All this is accomplished by sending these values and character kinematics as function parameters based on which we navigate the tree and call respective functions to evaluate the conditions as node values. At each node of the tree we check for only a subset of these values but all of them are made visible at every game loop since the decision trees are implemented as a combination of binary trees and other functions. One assumption that I made was that once we check a condition and obtain an action, no other conditions are checked, hence only one action is allowed at a time. I believe that the conditions must be checked in order of priority, for e.g. the collision with boundaries is checked before checking for speed, etc. The following diagram shows the above explained encapsulations.



For the next task, we had to implement the behavior tree structure to program a monster chasing the character. In order to differentiate between the two sprites, we color the monsterred. We make use of the data structure we implemented in the first part. In order to keep things simple, we let the character move the same as before, using the decision tree and define another tree for implementation of the behavior tree for the monster. Same as before, the character moves almost randomly and can be directed with mouse clicks if the user wants to change the final target point. The monster sprite moves at ¼ the speed of the character sprite to give it enough time to move away from it but since the character always moves towards the center after collision with the walls, the monster will always catch up to it. The behavior tree is also implemented using functions and binary tree data structure. The radius of satisfaction and deceleration are reduced for the monster character because it is already traveling at a reduced speed. Once collision occurs, detected by the tree by checking if the monster and character positions are the same, the “celebration” behavior is triggered. This results in the monster flashing at 5 different locations around the center of the room. To implement this a new variable to hold the value if celebration is needed is incorporated in the kinematic structure. Whenever the behavior tree makes it true, we complete this action. Same as before the tree needs the monster character kinematics as well as the character kinematics to make decisions. When the monster chases the character it will always implement the pathfinding algorithm. The behavior tree also triggers the reset behavior after celebration is complete. The behavior tree is also used

to trigger the monster on or off. Pressing the “m” key will activate the wander behavior and the monster will not target the character. Every “m” press will change the target wander location, otherwise it will keep moving towards the target set on the last “m” key press and teleport to the center of the screen whenever it goes out of bounds. No collisions are detected in this case. Pressing the “n” key will activate the monster and it will start the pursuit again by activating the path finding behavior. The following diagram shows the encapsulated data:



Given the parameters of explanation, the monster behaves as expected. Behavior trees involve a bit more complexity in comparison to the decision trees because they are somewhat of an extension of them.

Appendix

Monster (red) chasing the character (black).